

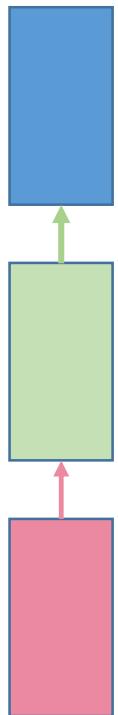
# Рекуррентные нейронные сети

# Как предсказать следующее слово?

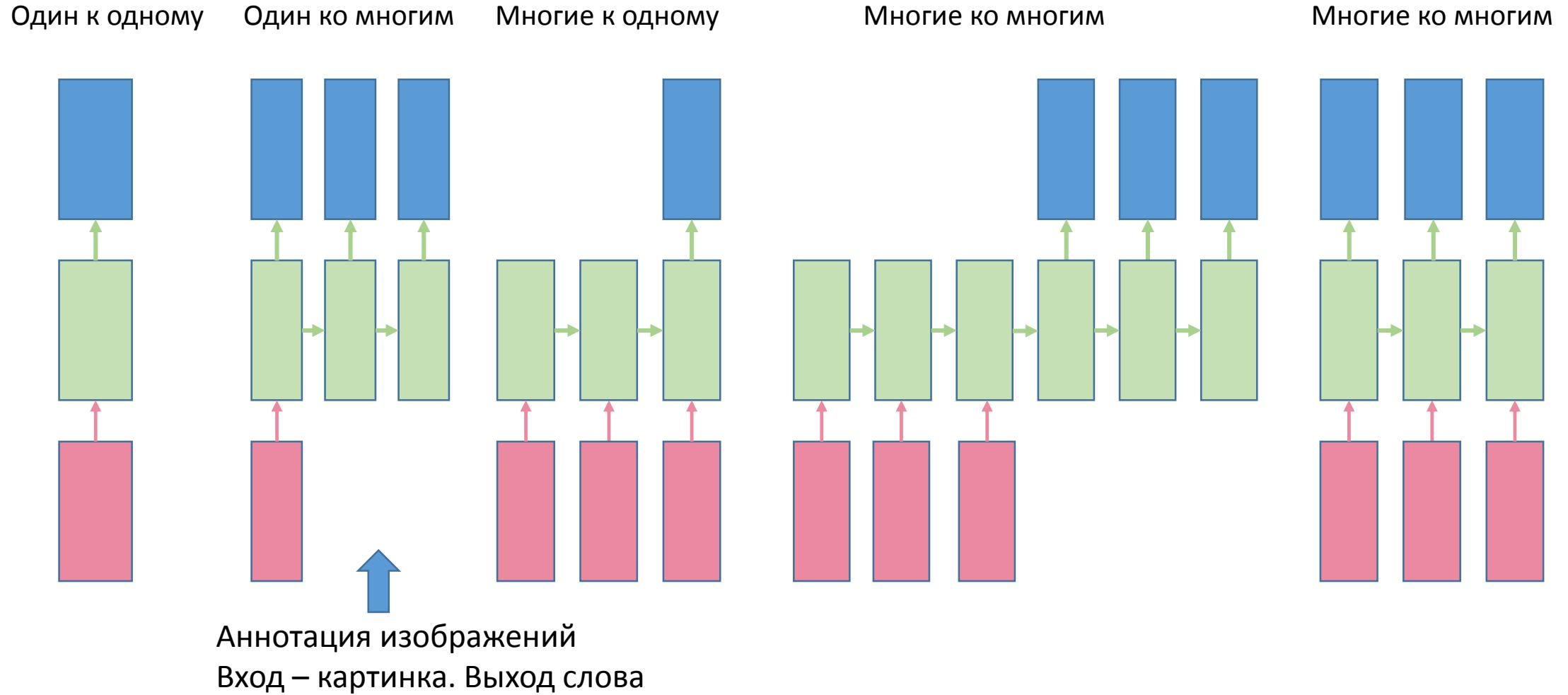
- Посмотреть на предыдущее слово?
- Посмотреть на предыдущие 3 слова?

# Сеть прямого распространения

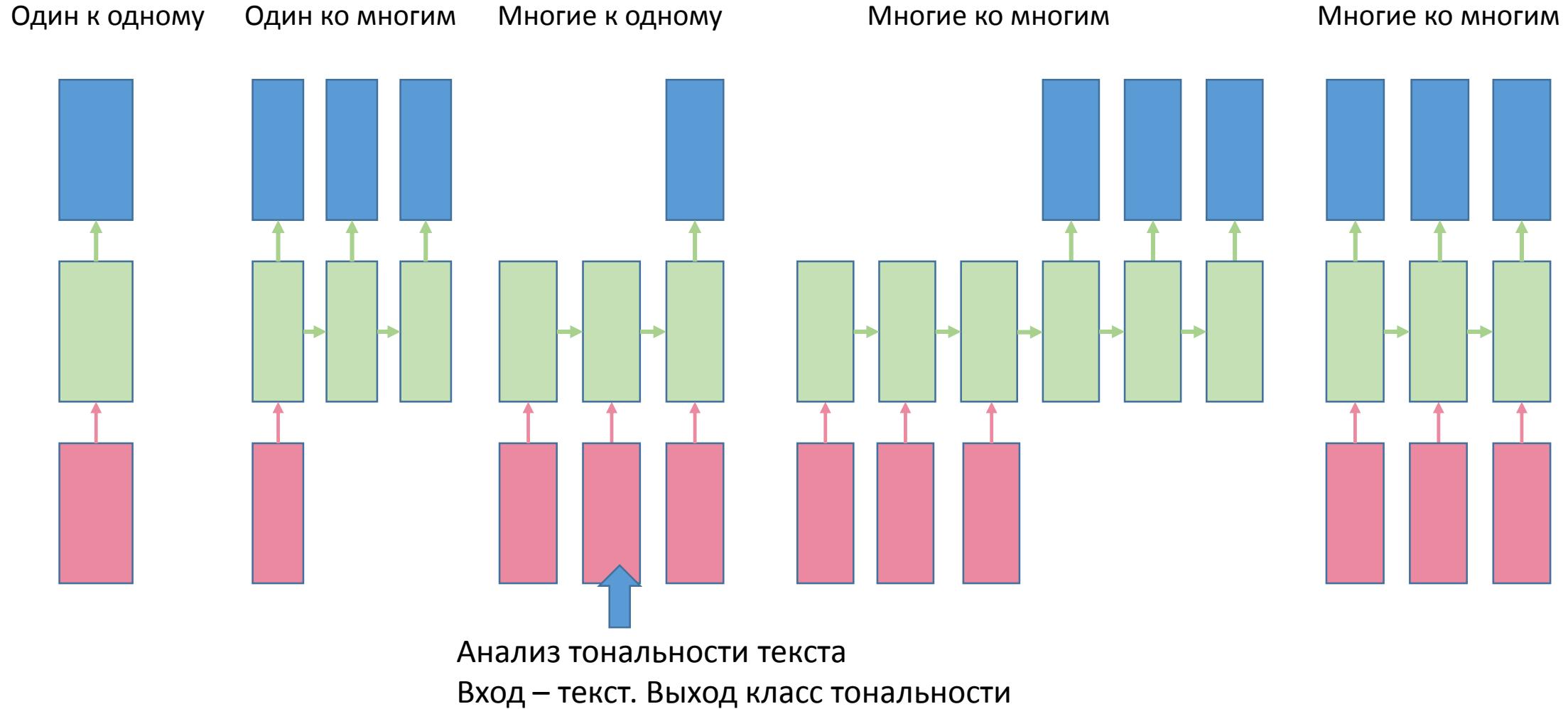
Один к одному



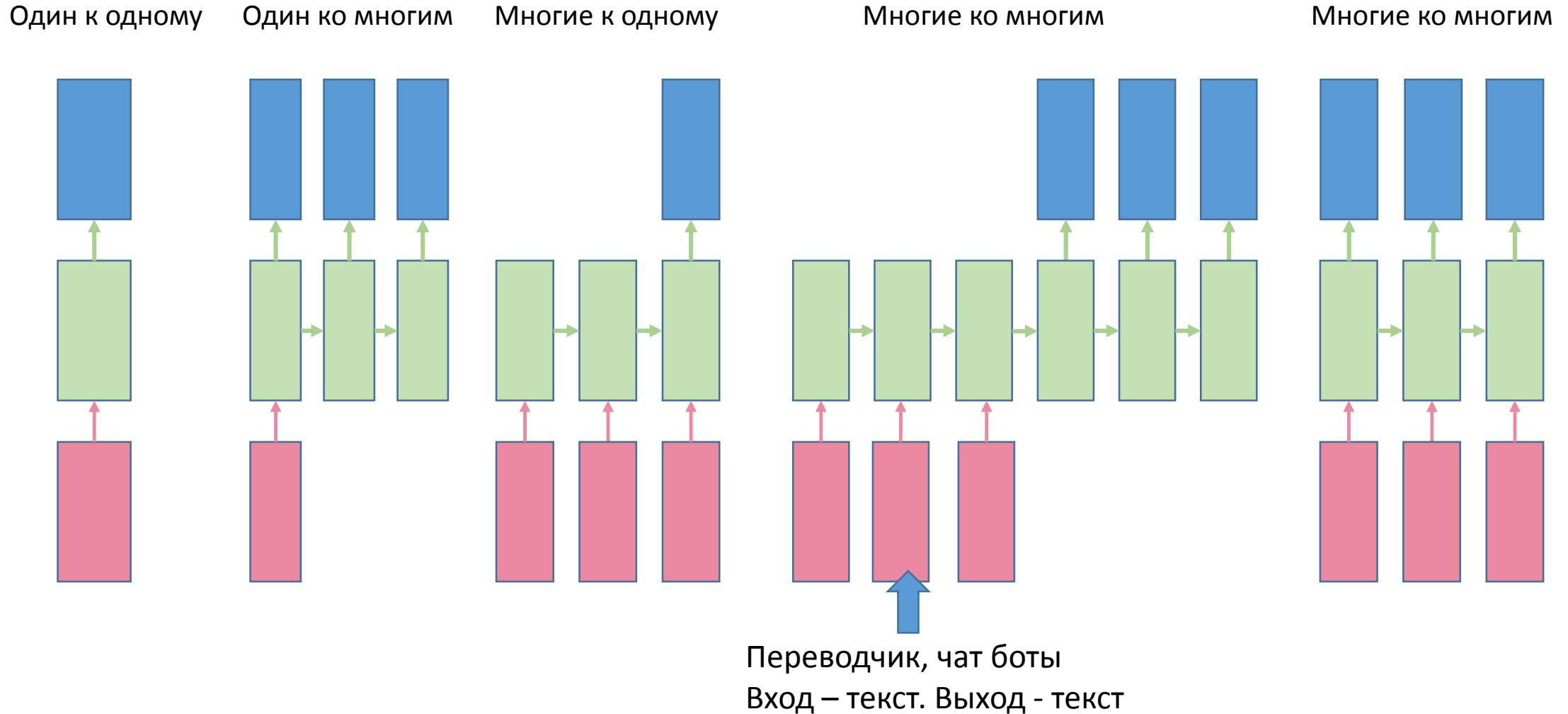
# Рекуррентная нейросеть



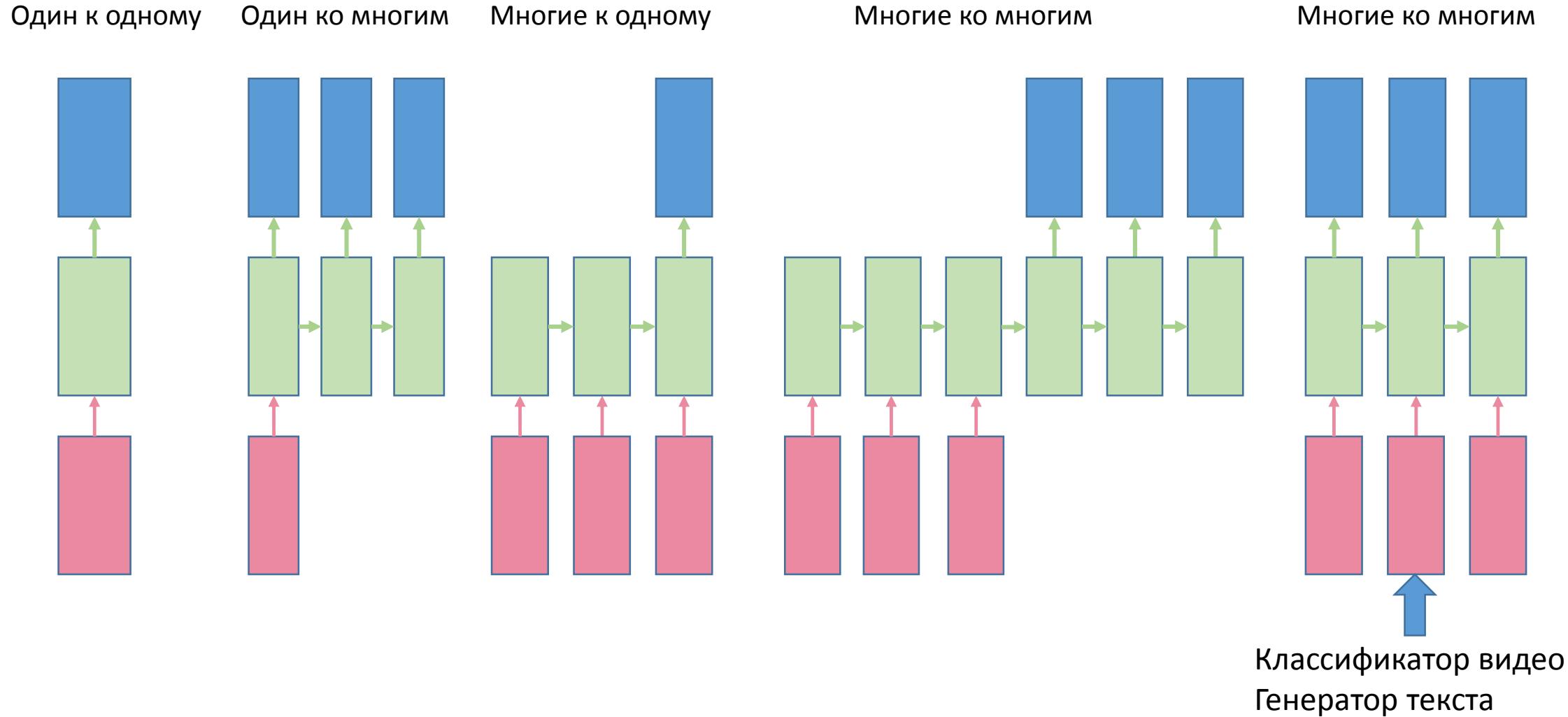
# Рекуррентная нейросеть



# Рекуррентная нейросеть



# Рекуррентная нейросеть



# Рекуррентная формула

Сеть подает на вход в момент времени  $t$  выход  $t-1$

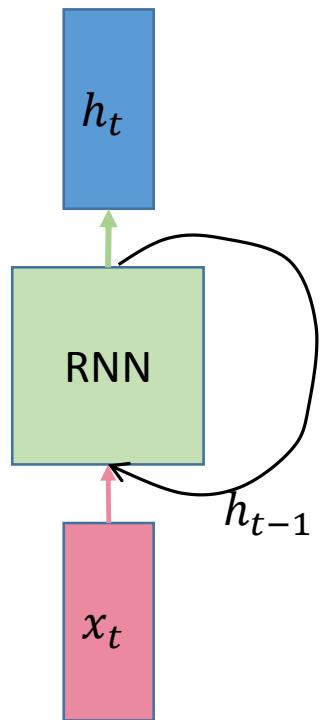
$$h_t = f_W(h_{t-1}, x_t)$$

Новое состояние

Старое состояние

Вход на шаге t

Функция с параметрами W

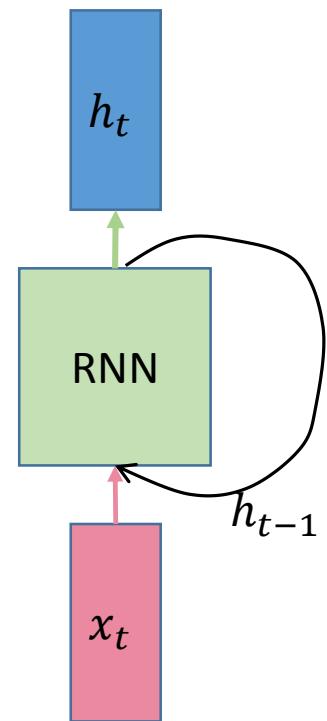


# Рекуррентная формула

Сеть подает на вход в момент времени  $t$  выход  $t-1$

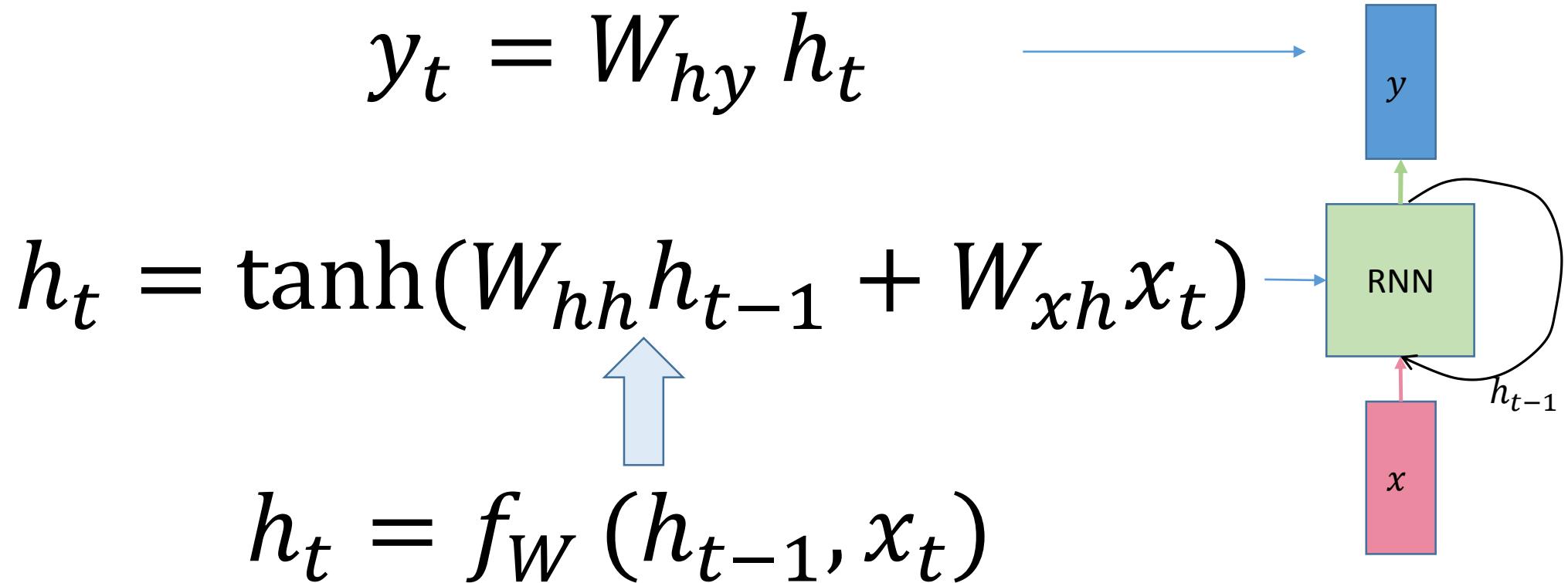
$$h_t = f_W(h_{t-1}, x_t)$$

Новое состояние  
Старое состояние  
Вход на шаге t  
Функция с параметрами W

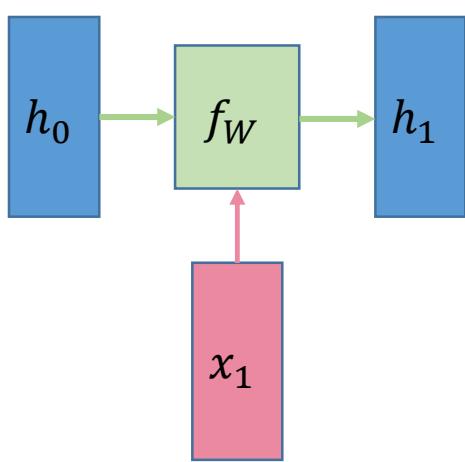


Для каждого момента времени используется  
одна функция и одна матрица весов

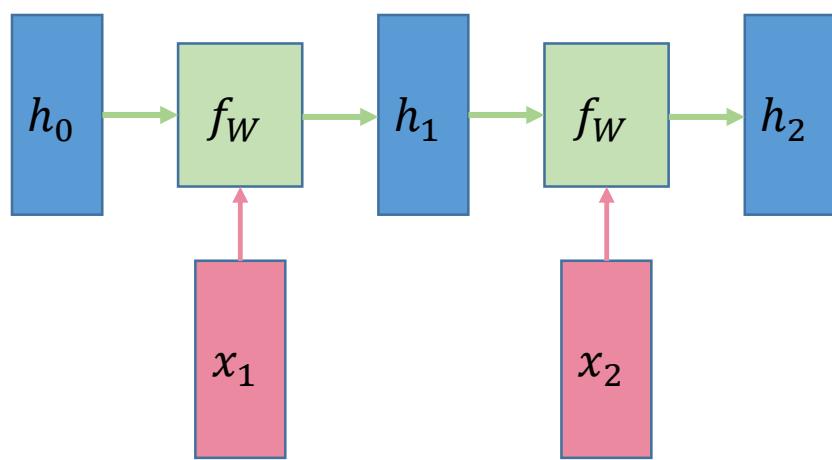
# Базовая рекуррентная сеть



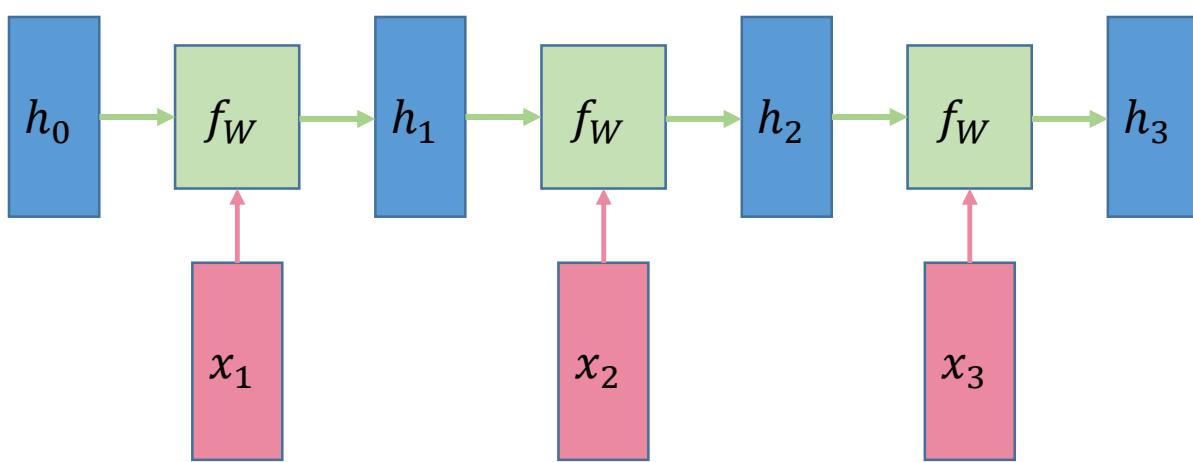
# RNN вычислительный граф



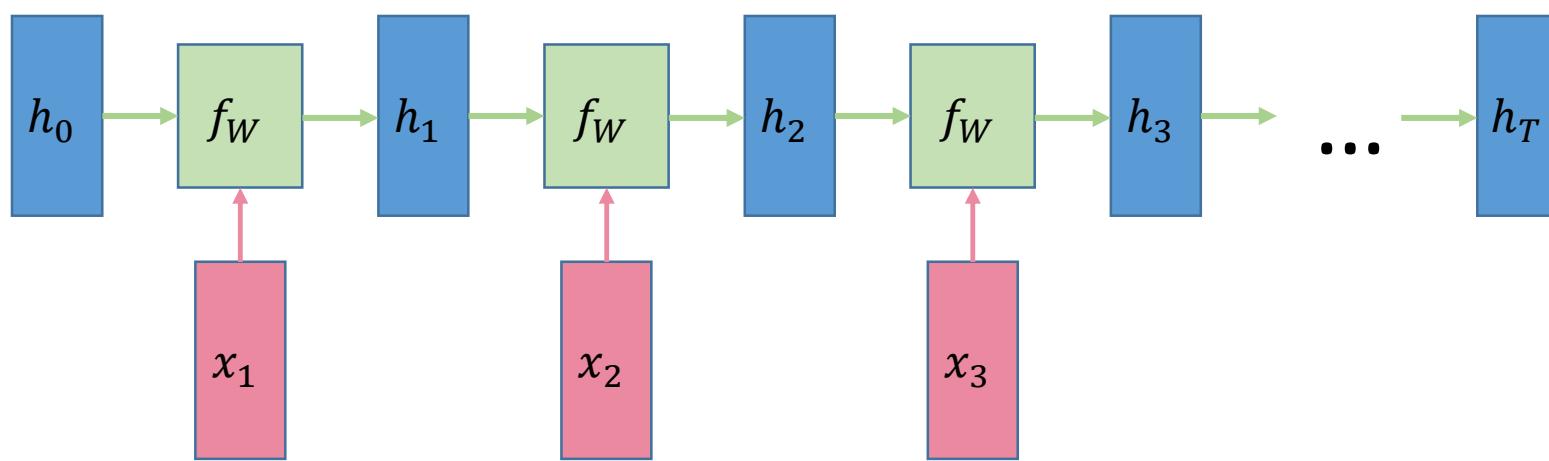
# RNN вычислительный граф



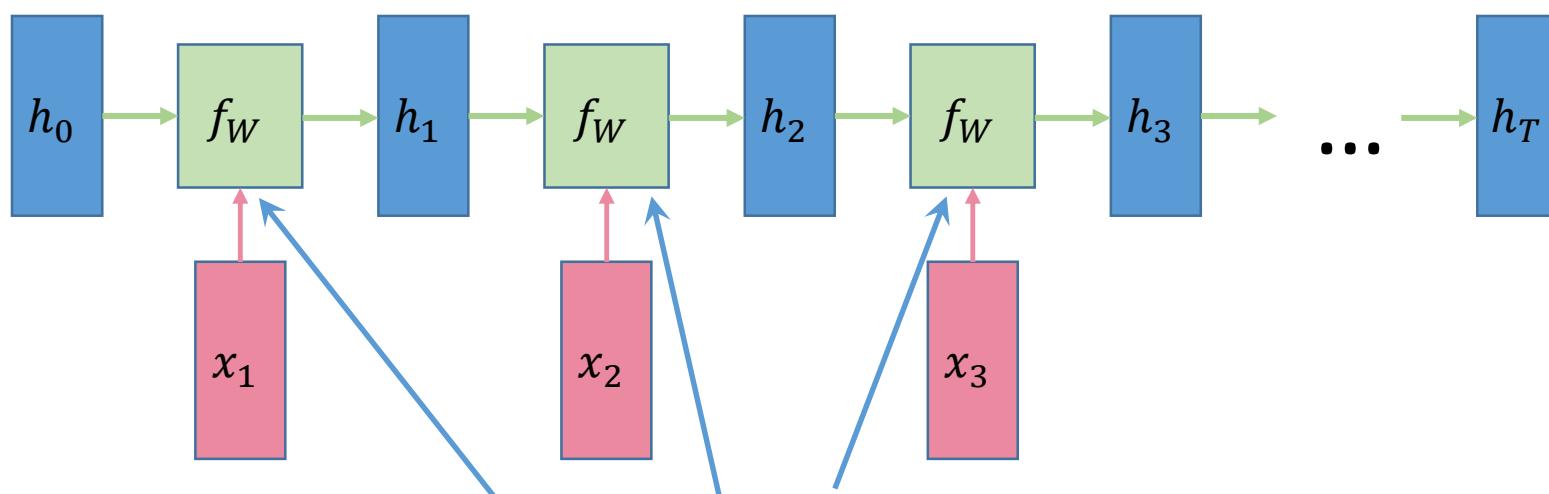
# RNN вычислительный граф



# RNN вычислительный граф

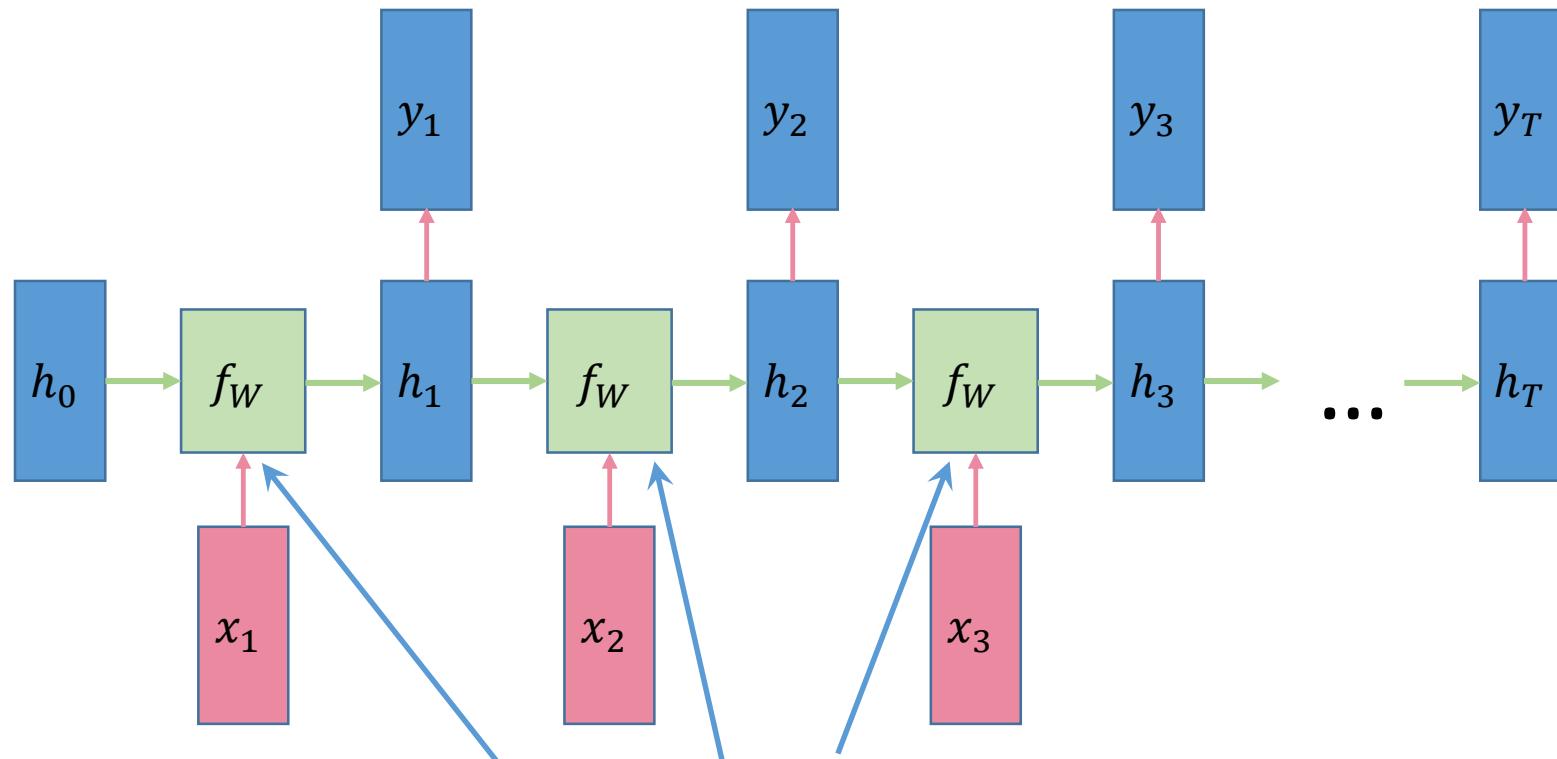


# RNN вычислительный граф



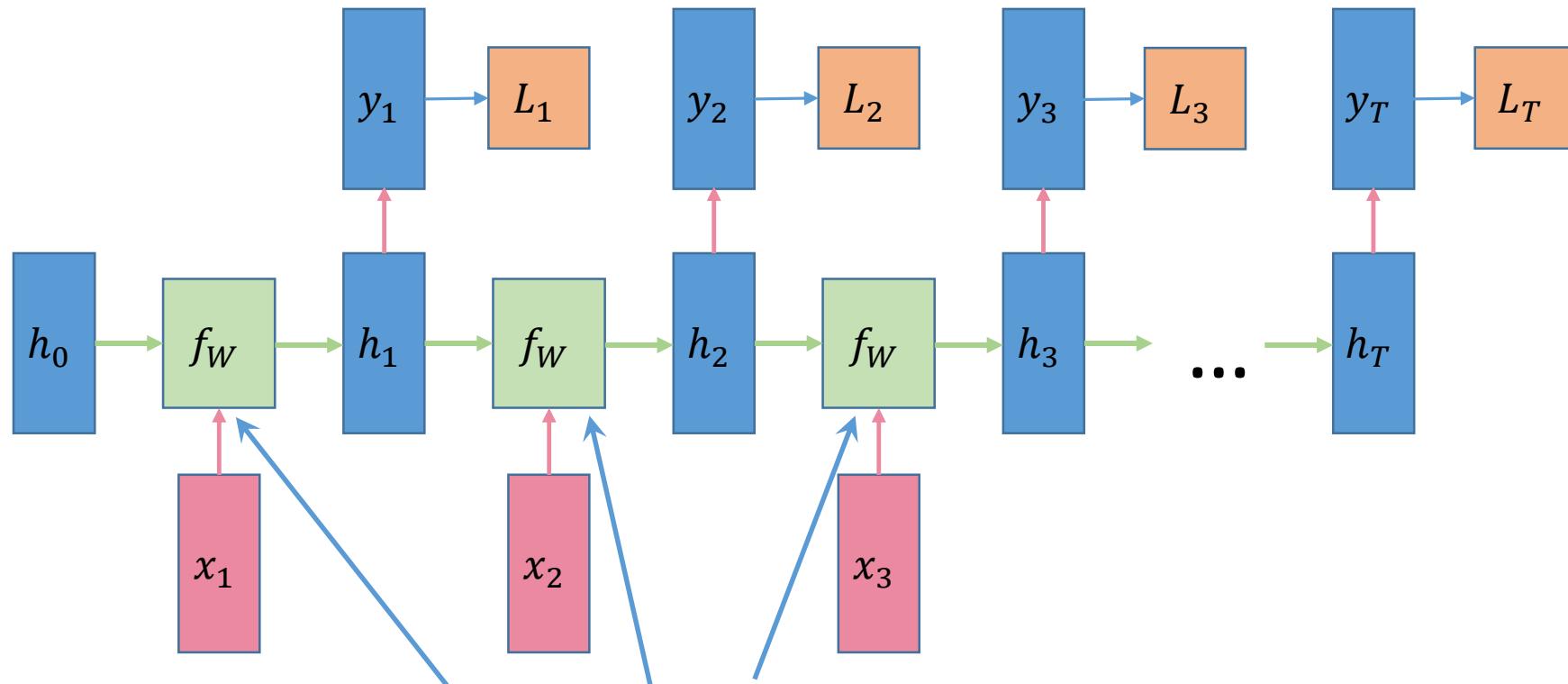
Одна матрица  $W$  на каждом шаге

# Вычислительный граф. Многие ко многим



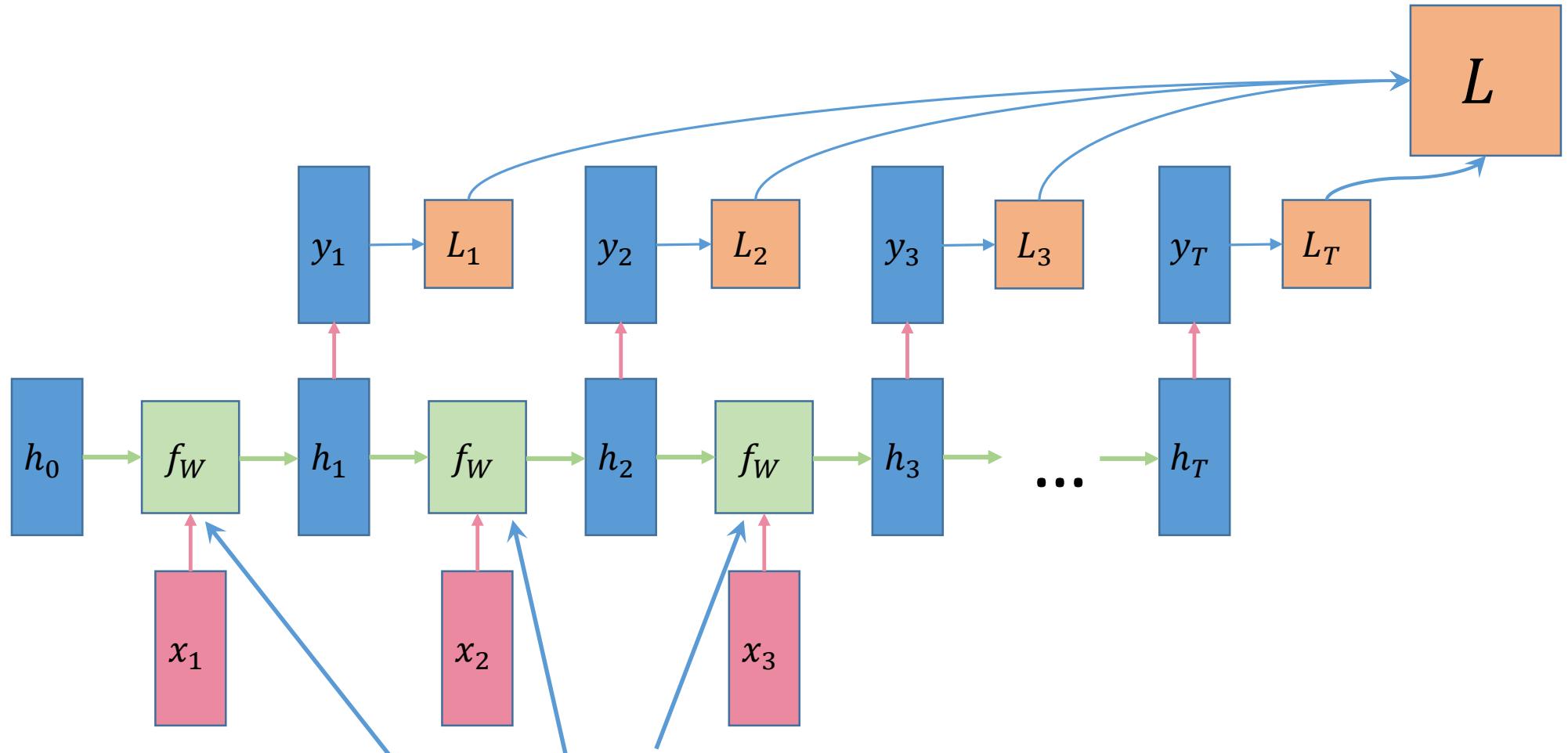
Одна матрица  $W$  на каждом шаге

# Вычислительный граф. Многие ко многим



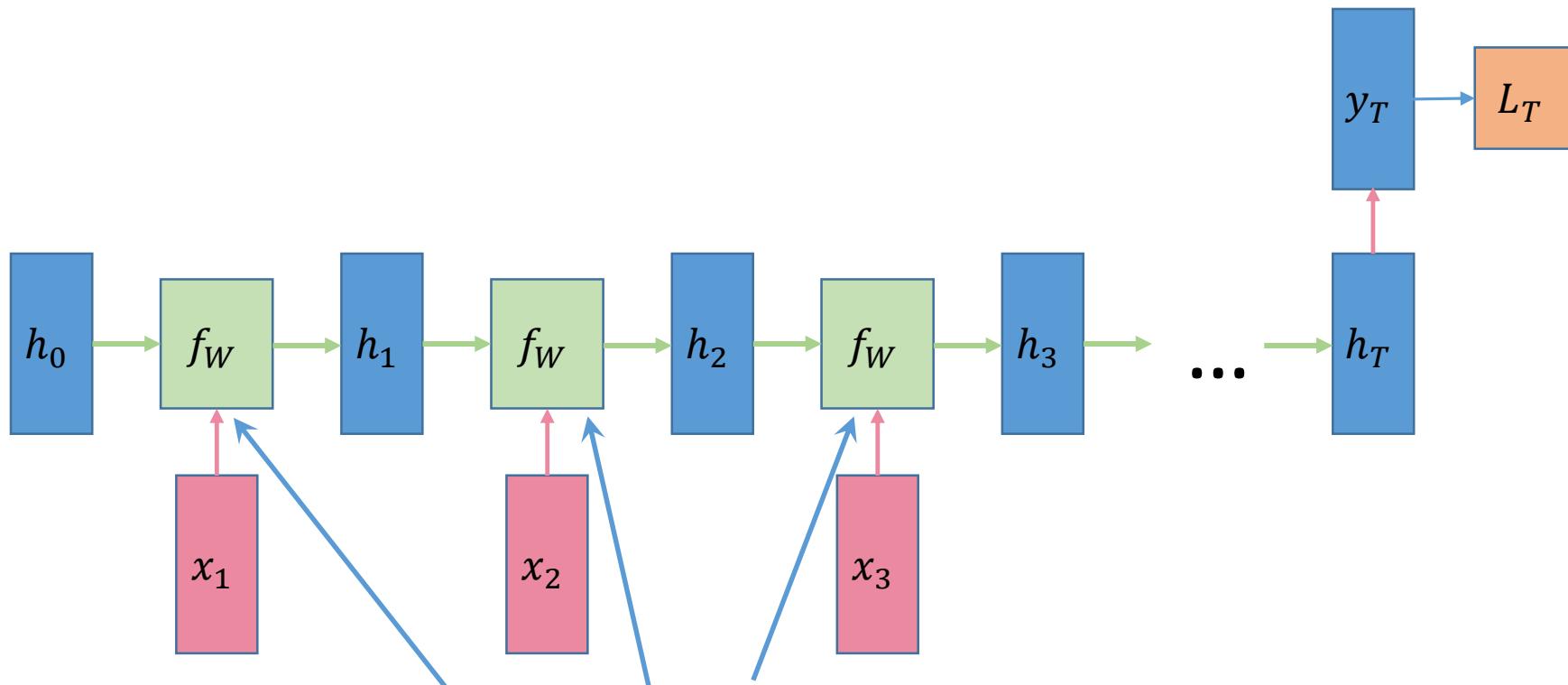
Одна матрица  $W$  на каждом шаге

# Вычислительный граф. Многие ко многим



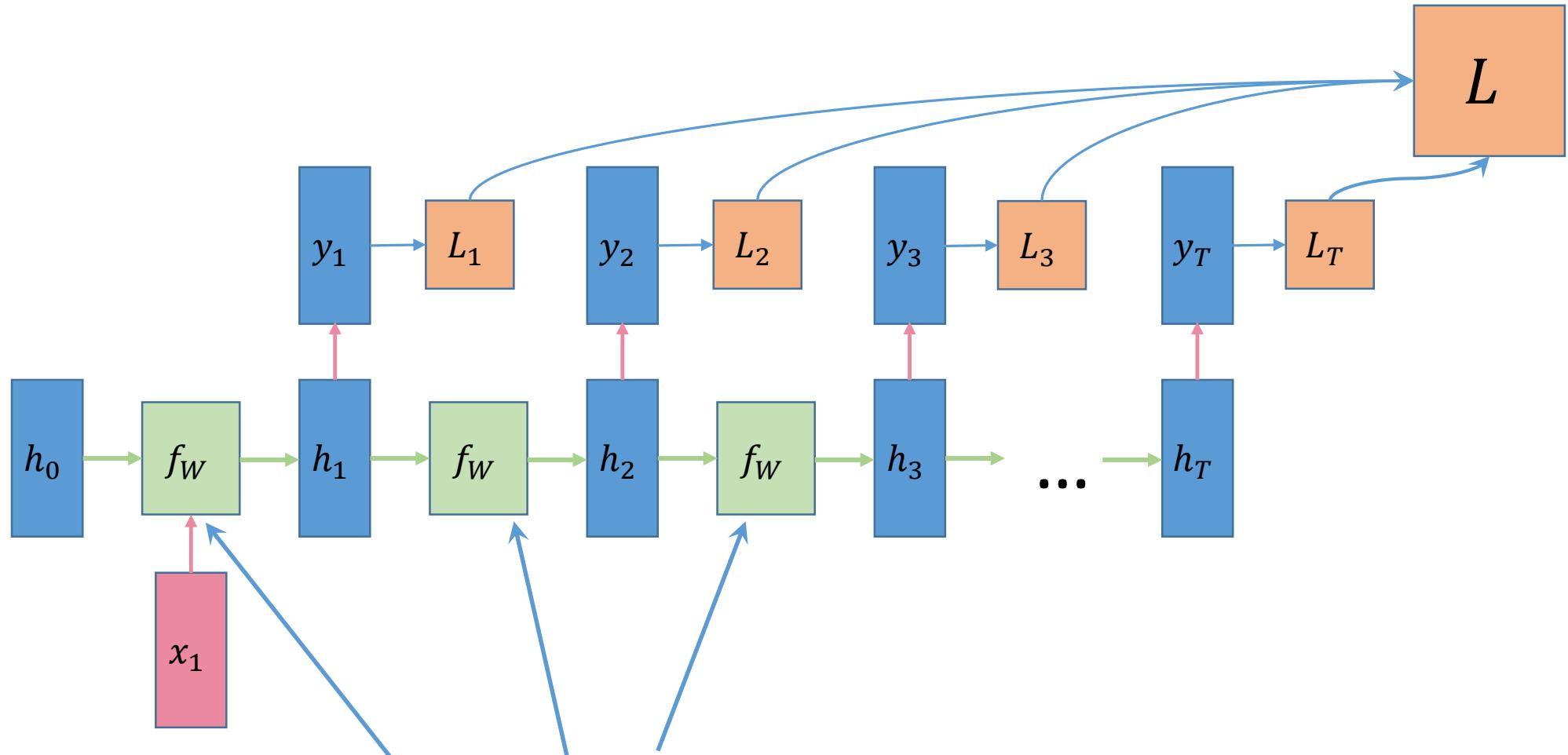
Одна матрица  $W$  на каждом шаге

# Вычислительный граф. Многие к одному



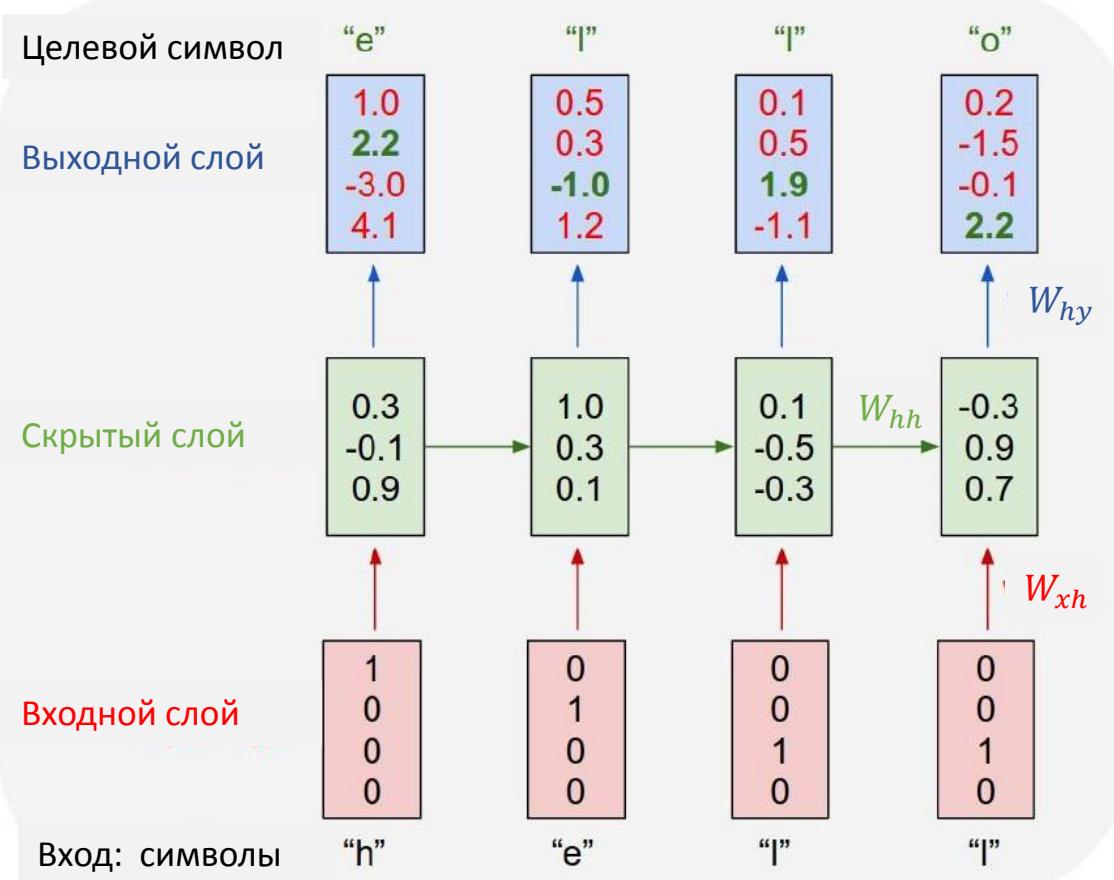
Одна матрица  $W$  на каждом шаге

# Вычислительный граф. Один ко многим



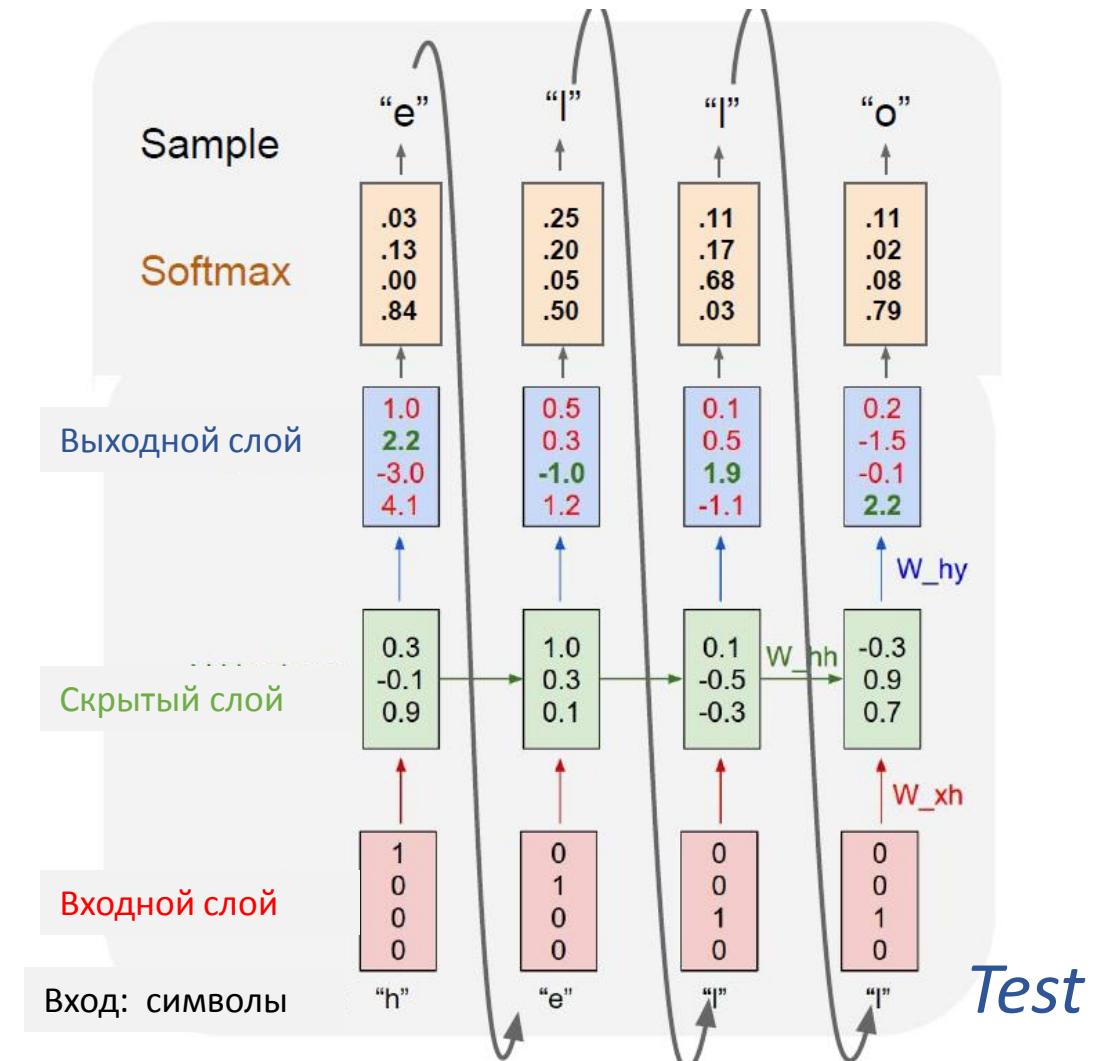
Одна матрица  $W$  на каждом шаге

# Character-level. Языковая модель

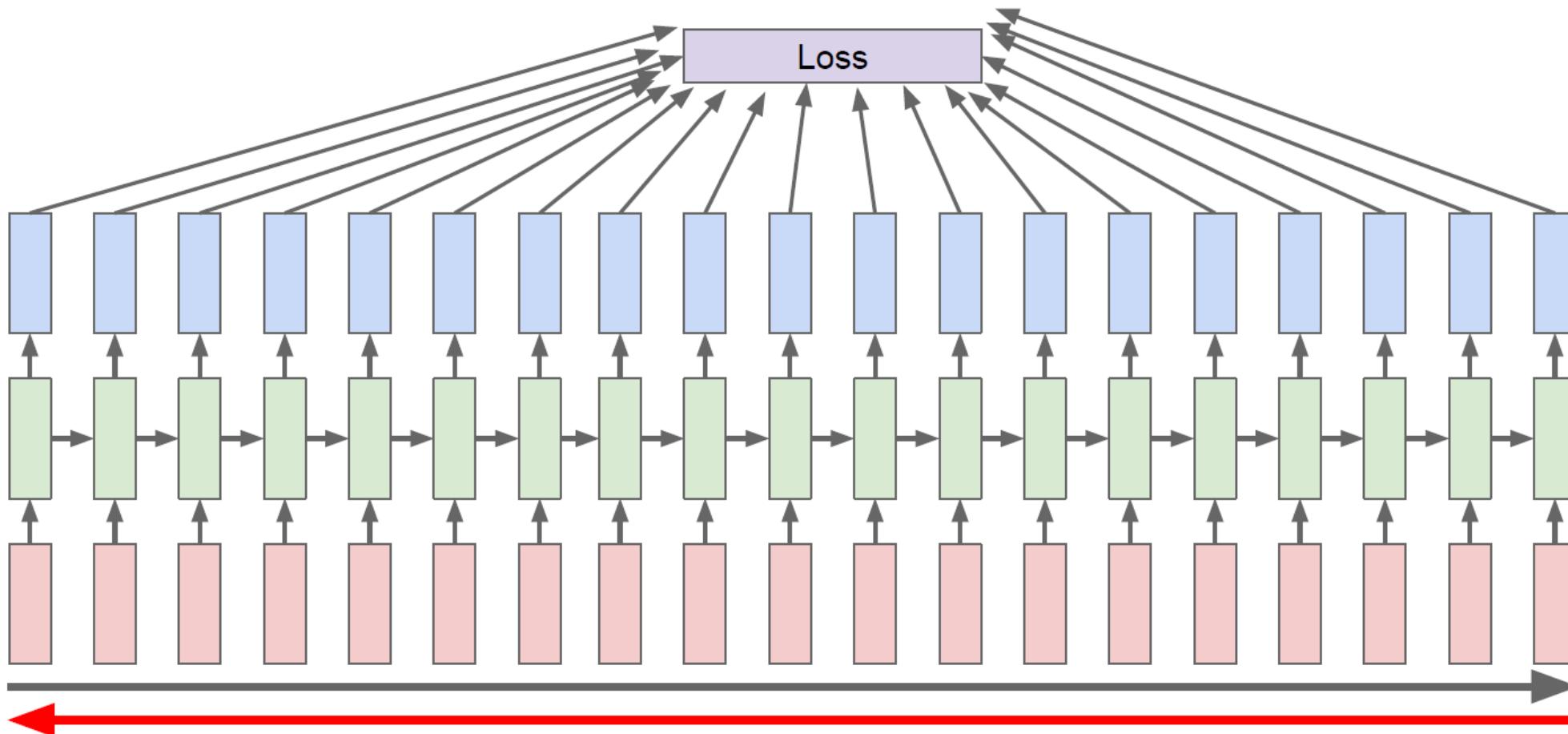


Train

Словарь: [h,e,l,o]

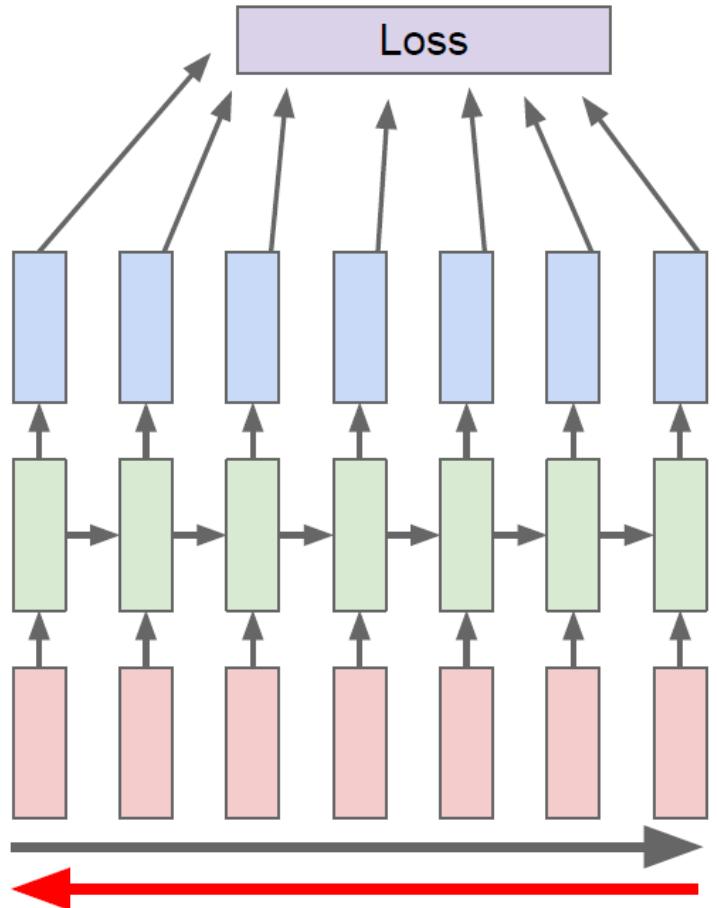


# Backpropagation по времени



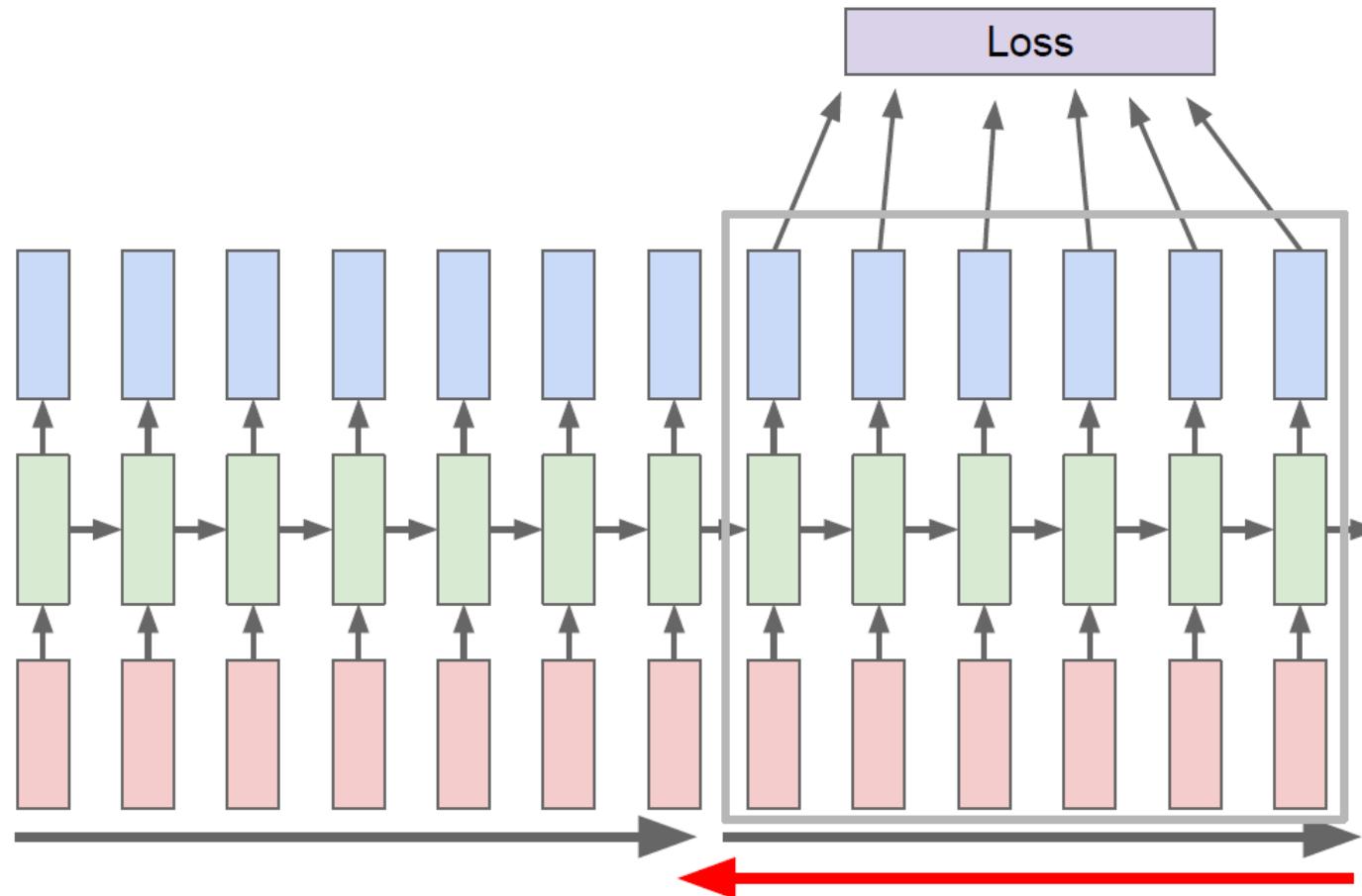
Прямой и обратный проход по всей последовательности

# Truncated Backpropagation по времени



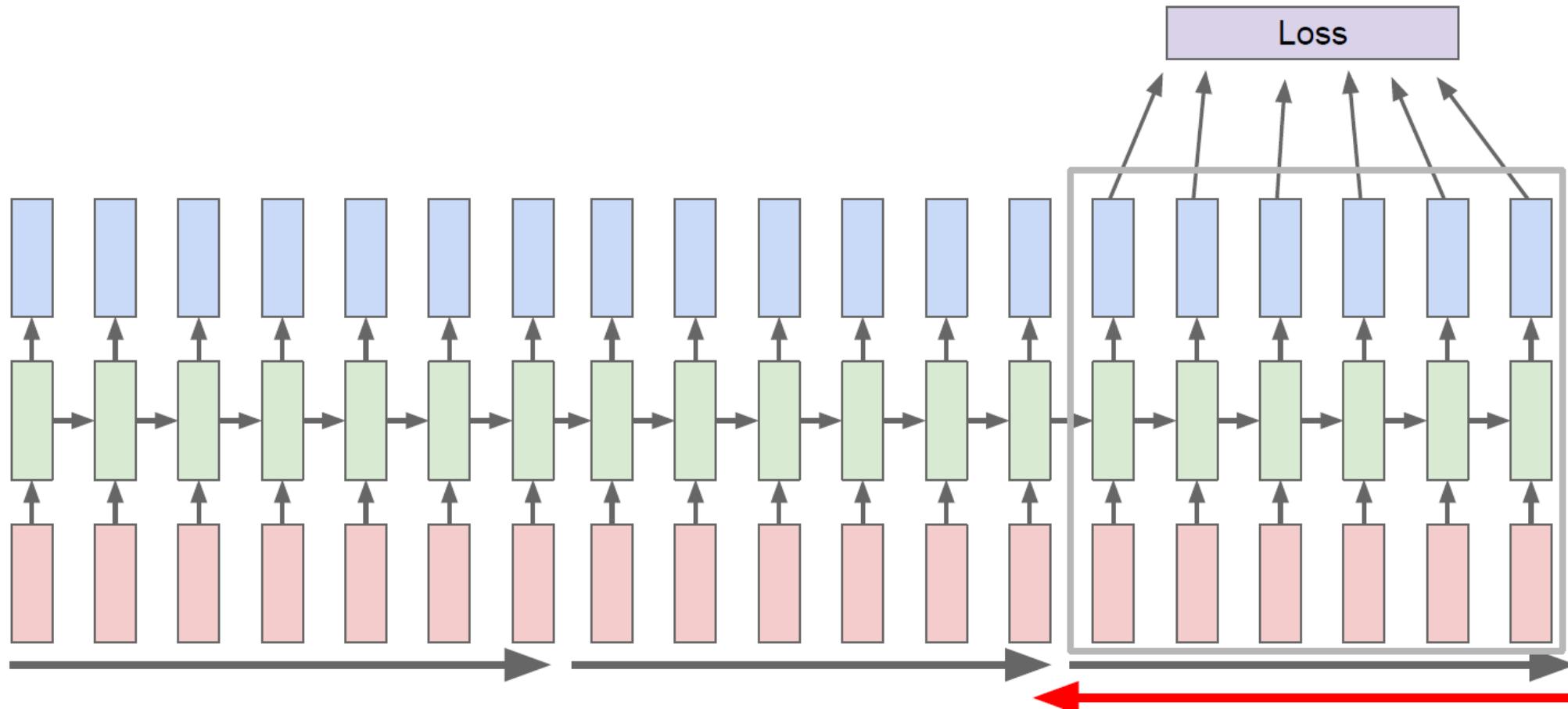
Прямой и обратный проход  
только по кусочку всей  
последовательности

# Truncated Backpropagation по времени



Сохраняем состояния  
предыдущего  
кусочка и делаем  
прямой и обратный  
проход по  
следующему

# Truncated Backpropagation по времени



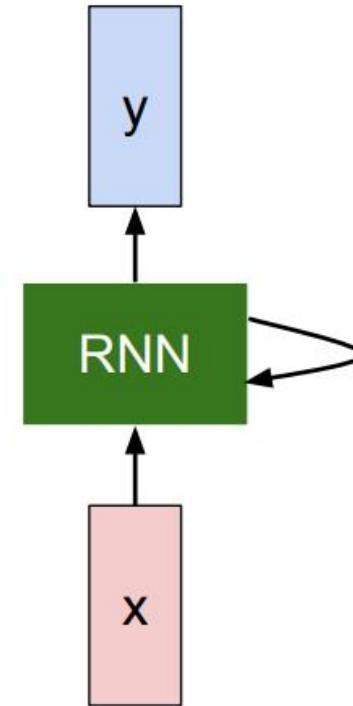
# Приимеры генерации текста

## THE SONNETS

by William Shakespeare

From fairest creatures we desire increase,  
That thereby beauty's rose might never die,  
But as the riper should by time decease,  
His tender heir might bear his memory:  
But thou, contracted to thine own bright eyes,  
Feed'st thy light's flame with self-substantial fuel,  
Making a famine where abundance lies,  
Thyself thy foe, to thy sweet self too cruel:  
Thou that art now the world's fresh ornament,  
And only herald to the gaudy spring,  
Within thine own bud buriest thy content,  
And tender churl mak'st waste in niggarding:  
Pity the world, or else this glutton be,  
To eat the world's due, by the grave and thee.

When forty winters shall besiege thy brow,  
And dig deep trenches in thy beauty's field,  
Thy youth's proud livery so gazed on now,  
Will be a tatter'd weed of small worth held:  
Then being asked, where all thy beauty lies,  
Where all the treasure of thy lusty days;  
To say, within thine own deep sunken eyes,  
Were an all-eating shame, and thriftless praise.  
How much more praise desp'rd thy beauty's use,  
If thou couldst answer 'This fair child of mine  
Shall sum my count, and make my old excuse,'  
Proving his beauty by succession thine!  
This were to be new made when thou art old,  
And see thy blood warm when thou feel'st it cold.



# Примеры генерации текста

at first:

tyntd-iafhatawiaoahrdemot lytdws e ,tfti, astai f ogoh eoase rrranbyne 'nhthnee e  
plia tkldrgd t o idoe ns,smtt h ne etie h,hregtrs nigtike,aoaenns lmg

↓ train more

"Tmont thithey" fomesscerliund  
Keushey. Thom here  
sheulke, anmerenith ol sivh I lalterthend Bleipile shuwy fil on aseterlome  
coaniogennc Phe lism thond hon at. MeiDimorotion in ther thize."

↓ train more

Aftair fall unsuch that the hall for Prince Velzonski's that me of  
her hearly, and behs to so arwage fiving were to it beloge, pavu say falling misfort  
how, and Gogition is so overelical and ofter.

↓ train more

"Why do what that day," replied Natasha, and wishing to himself the fact the  
princess, Princess Mary was easier, fed in had oftened him.  
Pierre aking his soul came to the packs and drove up his father-in-law women.

# Визуализация скрытого состояния

Text color corresponds to  $\tanh(c)$ , where -1 is red and +1 is blue.

Cell sensitive to position in line:

```
The sole importance of the crossing of the Berezina lies in the fact  
that it plainly and indubitably proved the fallacy of all the plans for  
cutting off the enemy's retreat and the soundness of the only possible  
line of action--the one Kutuzov and the general mass of the army  
demanded--namely, simply to follow the enemy up. The French crowd fled  
at a continually increasing speed and all its energy was directed to  
reaching its goal. It fled like a wounded animal and it was impossible  
to block its path. This was shown not so much by the arrangements it  
made for crossing as by what took place at the bridges. When the bridges  
broke down, unarmed soldiers, people from Moscow and women with children  
who were with the French transport, all--carried on by vis inertiae--  
pressed forward into boats and into the ice-covered water and did not,  
surrender.
```

Cell that turns on inside quotes:

```
"You mean to imply that I have nothing to eat out of.... On the  
contrary, I can supply you with everything even if you want to give  
dinner parties," warmly replied Chichagov, who tried by every word he  
spoke to prove his own rectitude and therefore imagined Kutuzov to be  
animated by the same desire.
```

```
Kutuzov, shrugging his shoulders, replied with his subtle penetrating  
smile: "I meant merely to say what I said."
```

Cell that robustly activates inside if statements:

```
static int __dequeue_signal(struct sigpending *pending, sigset_t *mask,  
    siginfo_t *info)  
{  
    int sig = next_signal(pending, mask);  
    if (sig) {  
        if (current->notifier) {  
            if (sigismember(current->notifier_mask, sig)) {  
                if (!!(current->notifier)(current->notifier_data)) {  
                    clear_thread_flag(TIF_SIGPENDING);  
                    return 0;  
                }  
            }  
            collect_signal(sig, pending, info);  
        }  
    }  
    return sig;  
}
```

A large portion of cells are not easily interpretable. Here is a typical example:

```
/* Unpack a filter field's string representation from user-space  
 * buffer. */  
char *audit_unpack_string(void **bufp, size_t *remain, size_t len)  
{  
    char *str;  
    if (!*bufp || (len == 0) || (len > *remain))  
        return ERR_PTR(-EINVAL);  
    /* Of the currently implemented string fields, PATH_MAX  
     * defines the longest valid length.  
     */
```

Cell that turns on inside comments and quotes:

```
/* Duplicate LSM field information. The lsm_rule is opaque, so  
 * we re-initialized it */  
static inline int audit_dupe_lsm_field(struct audit_field *df,  
    struct Audit_field *sf)  
{  
    int ret = 0;  
    char *lsm_str;  
    /* our own copy of lsm_str */  
    lsm_str = kstrdup(sf->lsm_str, GFP_KERNEL);  
    if (unlikely(!lsm_str))  
        return -ENOMEM;  
    df->lsm_str = lsm_str;  
    /* our own (refreshed) copy of lsm_rule */  
    ret = security_audit_rule_init(df->type, df->op, df->lsm_str,  
        (void *)df->lsm_rule);  
    /* Keep currently invalid fields around in case they  
     * become valid after a policy reload. */  
    if (ret == -EINVAL)  
        pr_warn("audit rule for LSM \\\"%s\\\" is invalid\\n",  
            df->lsm_str);  
    ret = 0;  
  
    return ret;
```

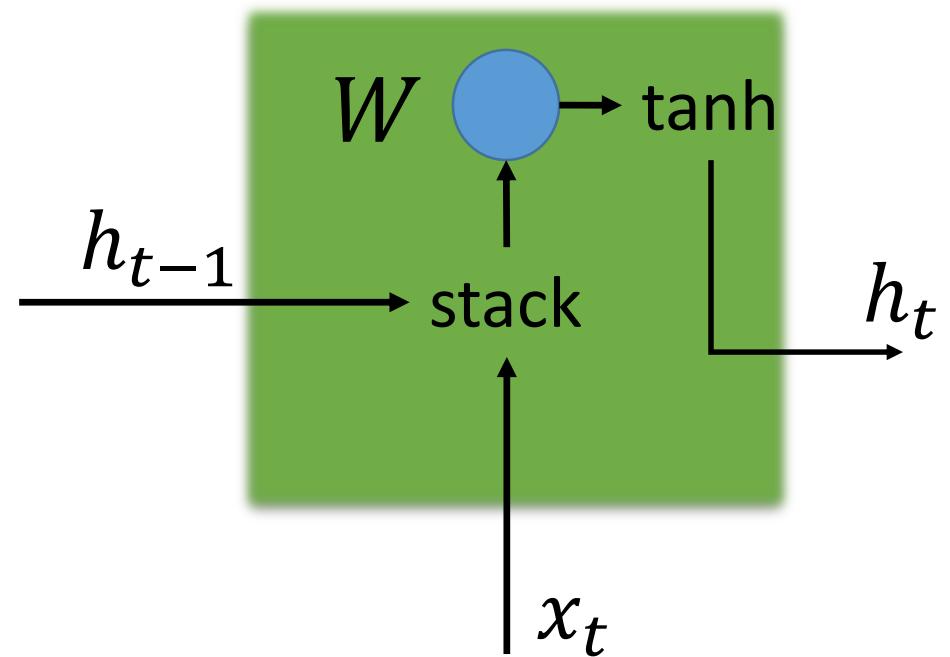
Cell that is sensitive to the depth of an expression:

```
#ifdef CONFIG_AUDITSYSCALL  
static inline int audit_match_class_bits(int class, u32 *mask)  
{  
    int i;  
    if (classes[class]) {  
        for (i = 0; i < AUDIT_BITMASK_SIZE; i++)  
            if (mask[i] & classes[class][i])  
                return 0;  
    }  
    return 1;  
}
```

Cell that might be helpful in predicting a new line. Note that it only turns on for some ")":

```
char *audit_unpack_string(void **bufp, size_t *remain, size_t len)  
{  
    char *str;  
    if (!*bufp || (len == 0) || (len > *remain))  
        return ERR_PTR(-EINVAL);  
    /* Of the currently implemented string fields, PATH_MAX  
     * defines the longest valid length.  
     */  
    if (len > PATH_MAX)  
        return ERR_PTR(-ENAMETOOLONG);  
    str = kmalloc(len + 1, GFP_KERNEL);  
    if (unlikely(!str))  
        return ERR_PTR(-ENOMEM);  
    memcpy(str, *bufp, len);  
    str[len] = 0;  
    *bufp += len;  
    *remain -= len;  
    return str;
```

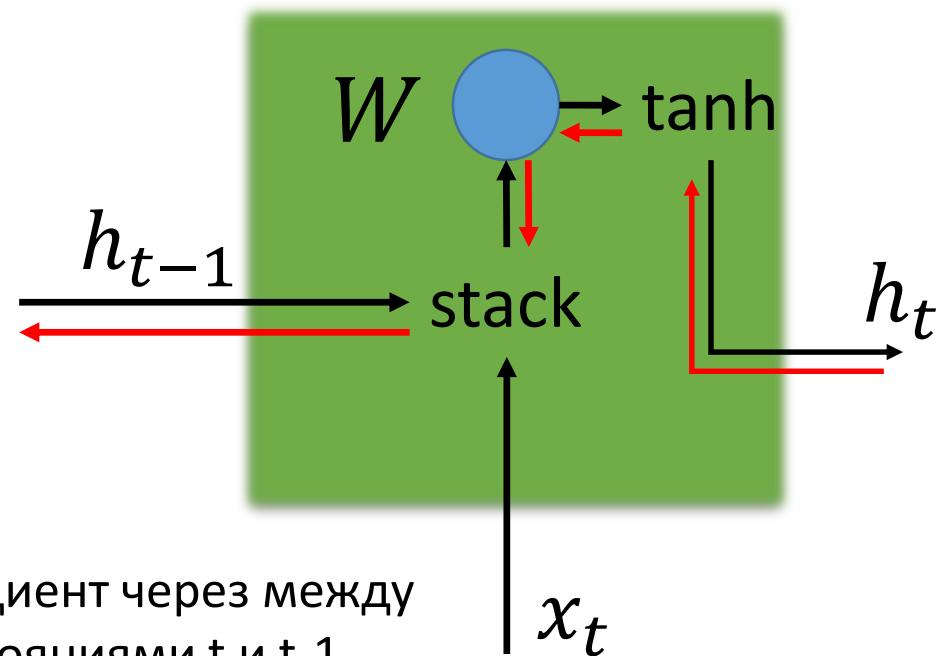
# Градиент через RNN



$$\begin{aligned} h_t &= \tanh(W_{hh}h_{t-1} + W_{xh}x_t) \\ &= \tanh((W_{hh} \quad W_{xh}) \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}) \\ &= \tanh( W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}) \end{aligned}$$

Bengio et al, “Learning long-term dependencies with gradient descent is difficult”, IEEE Transactions on Neural Networks, 1994  
Pascanu et al, “On the difficulty of training recurrent neural networks”, ICML 2013

# Градиент через RNN

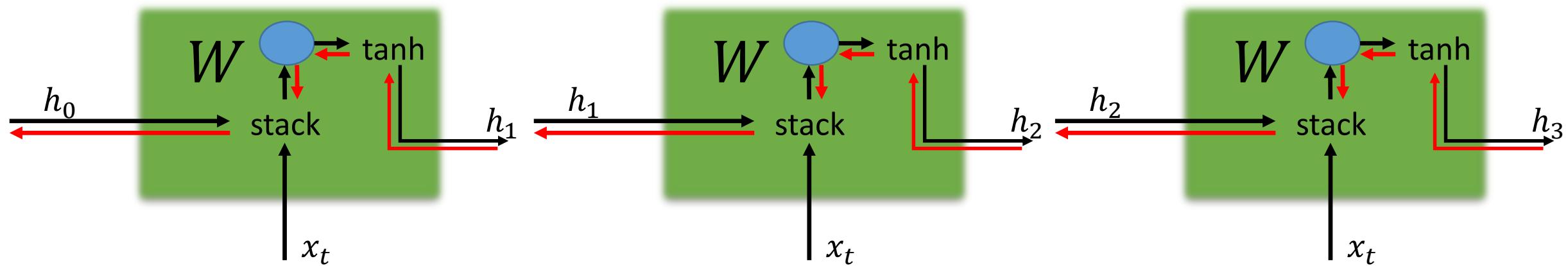


Градиент через между  
состояниями  $t$  и  $t-1$   
умножается на матрицу  
весов  $W^T$

$$\begin{aligned} h_t &= \tanh(W_{hh}h_{t-1} + W_{xh}x_t) \\ &= \tanh((W_{hh} \quad W_{xh}) \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}) \\ &= \tanh( W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}) \end{aligned}$$

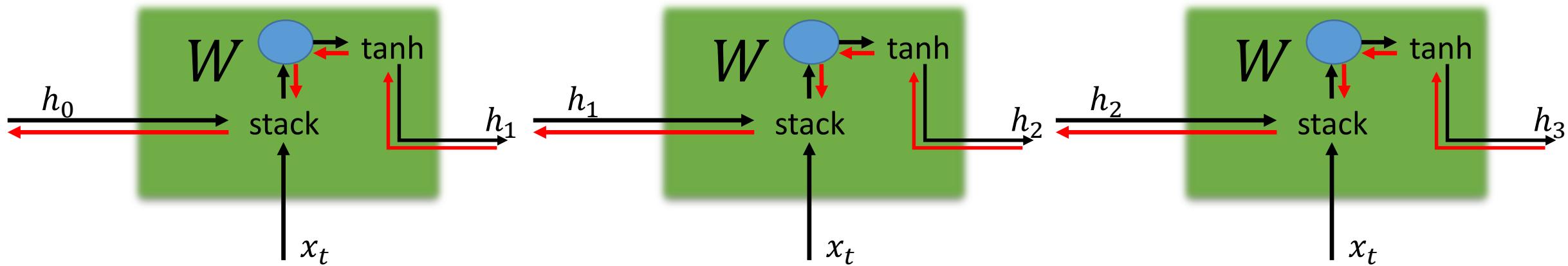
Bengio et al, “Learning long-term dependencies with gradient descent is difficult”, IEEE Transactions on Neural Networks, 1994  
Pascanu et al, “On the difficulty of training recurrent neural networks”, ICML 2013

# Градиент через RNN



Вычисление градиента для состояния 0 будет приводить к многократному умножению на матрицу весов  $W$

# Градиент через RNN



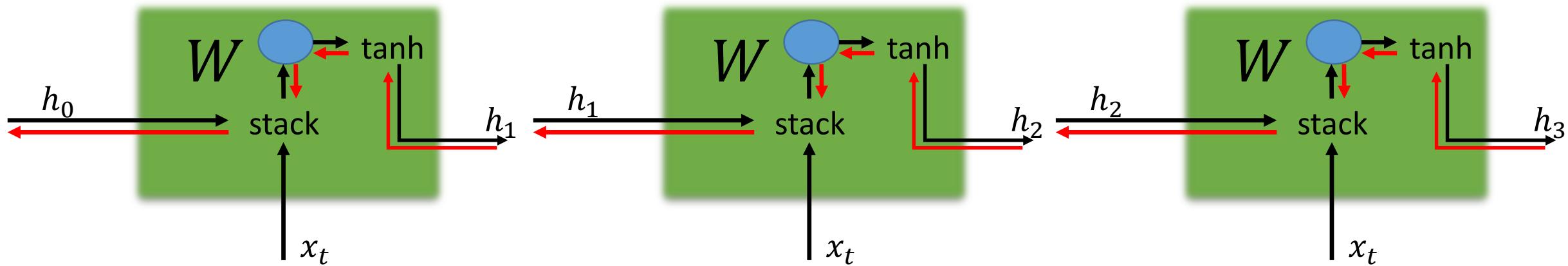
Вычисление градиента для состояния 0 будет приводить к многократному умножению на матрицу весов  $W$

Значения матрицы больше 1:

**Exploding gradient**

Значения матрицы меньше 1:  
**Vanishing gradient**

# Градиент через RNN



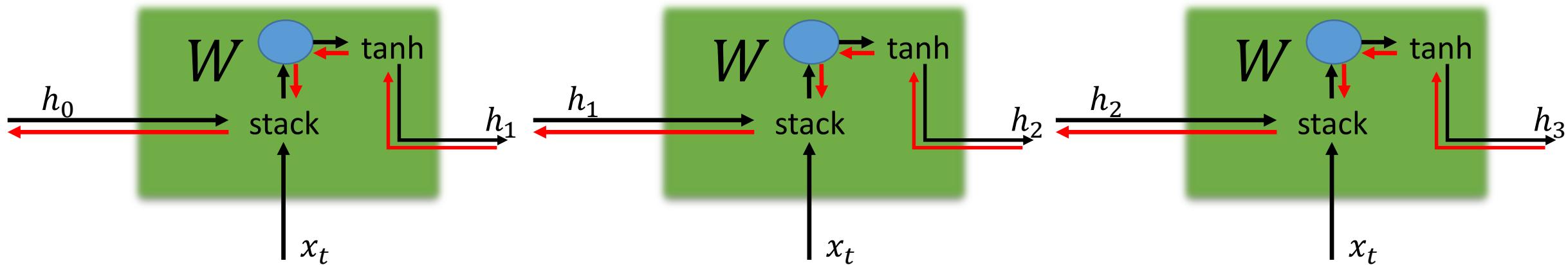
Вычисление градиента для состояния 0 будет приводить к многократному умножению на матрицу весов  $W$

Значения матрицы больше 1:  
**Exploding gradient**



```
torch.nn.utils.clip_grad_norm_(model.parameters(), 1)
```

# Градиент через RNN



Вычисление градиента для состояния 0 будет приводить к многократному умножению на матрицу весов  $W$

Значения матрицы меньше 1:  
**Vanishing gradient**

Нужно менять архитектуру RNN

# Проблемы RNN сетей

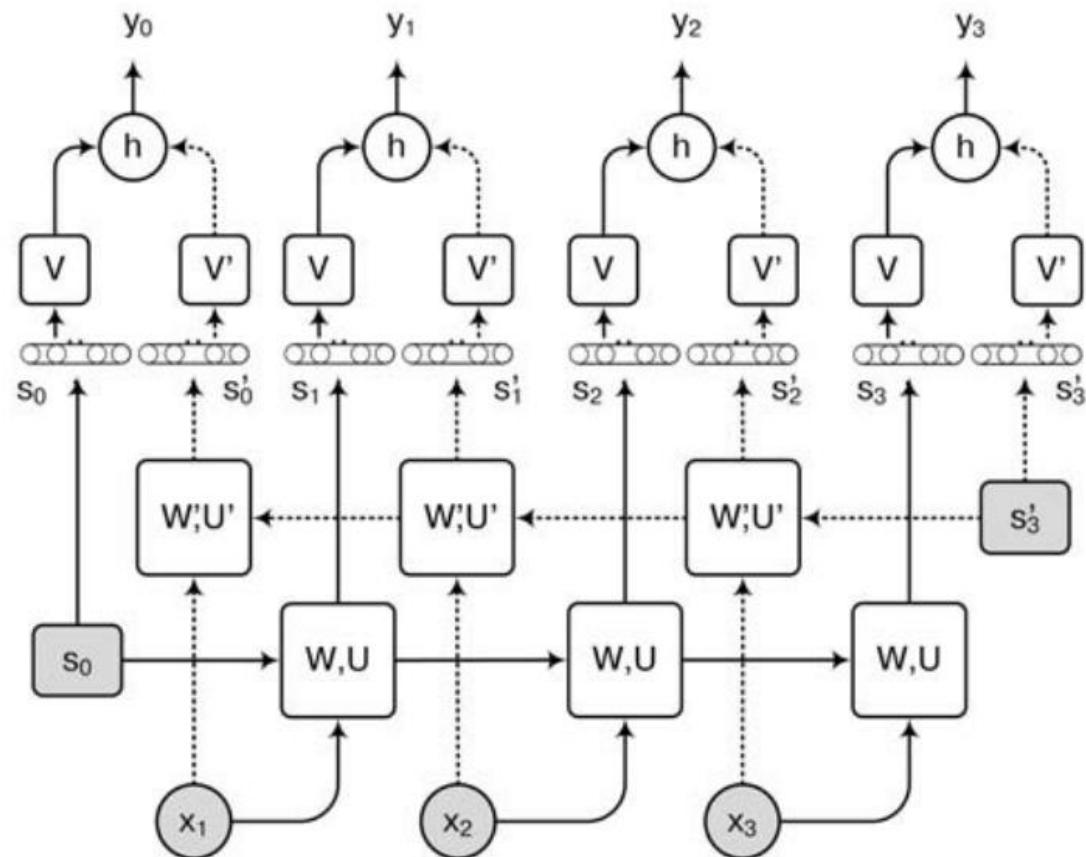
**Не может видеть вперед**

Решает при помощи **bidirectional RNN**.

Считаем два **hidden\_state**

слева-направо и

справа-налево



$$\begin{aligned} s_t &= \sigma(\mathbf{b} + Ws_{t-1} + Ux_t), & s'_t &= \sigma(\mathbf{b}' + W's'_{t+1} + U'x_t), \\ o_t &= \mathbf{c} + Vs_t + V's'_t, & y_t &= h(o_t). \end{aligned}$$

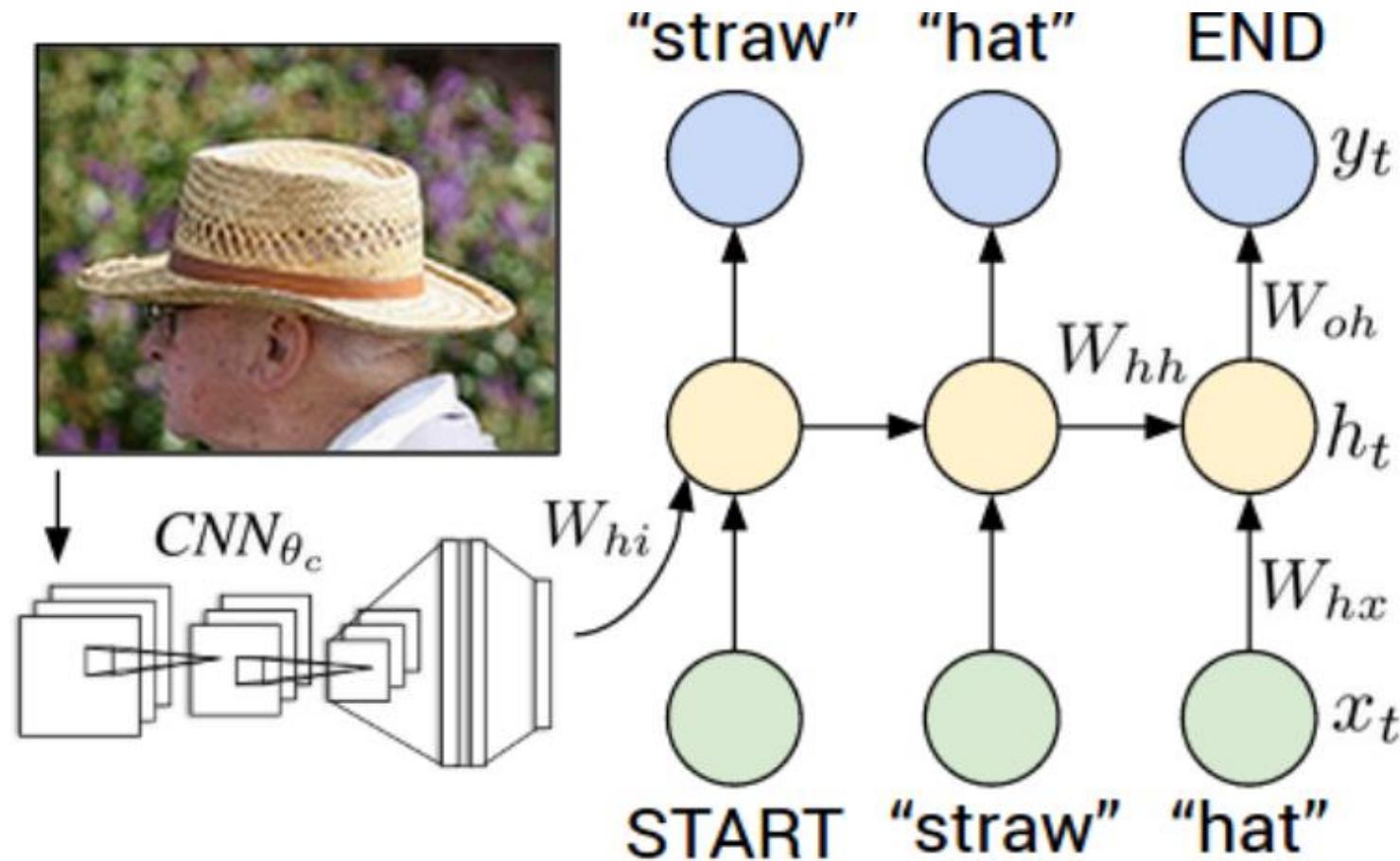
# Проблемы RNN сетей

Не может видеть далеко назад - при прохождении градиента обратно – происходит экспоненциальное затухание градиента, чем дальше назад – тем больше затухает.

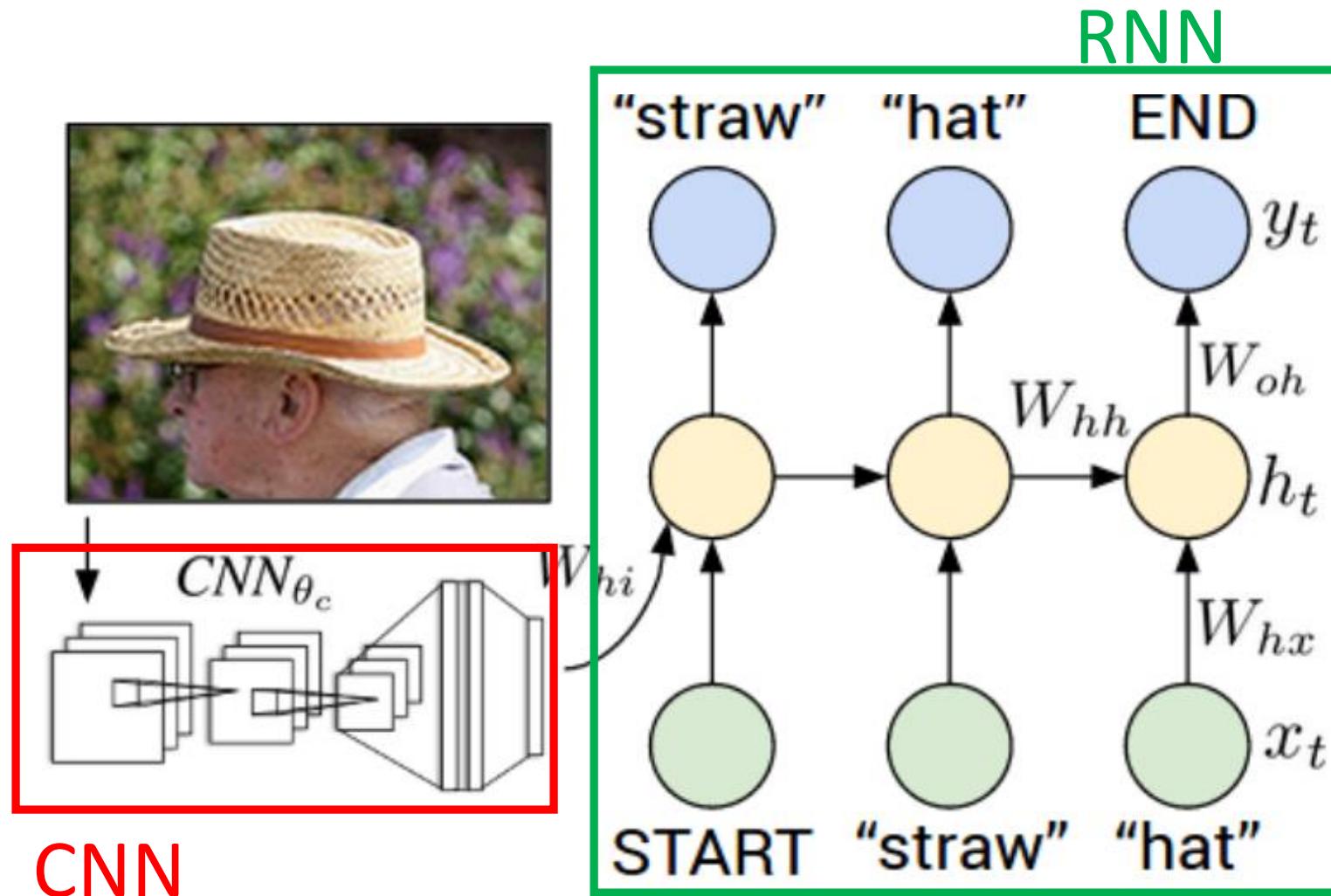
Решение - **LSTM**

# Приложения RNN сетей

# Image Captioning

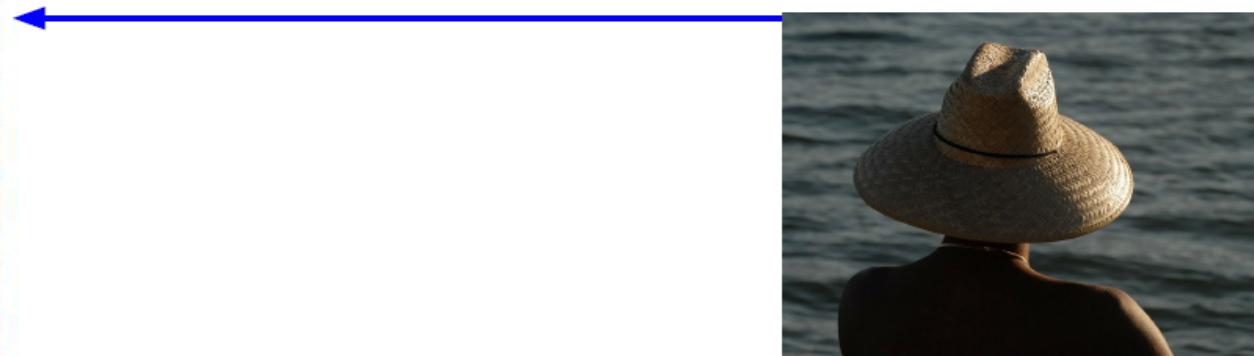


# Image Captioning

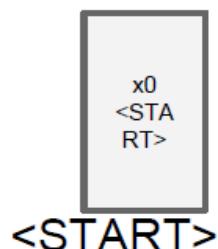




test image

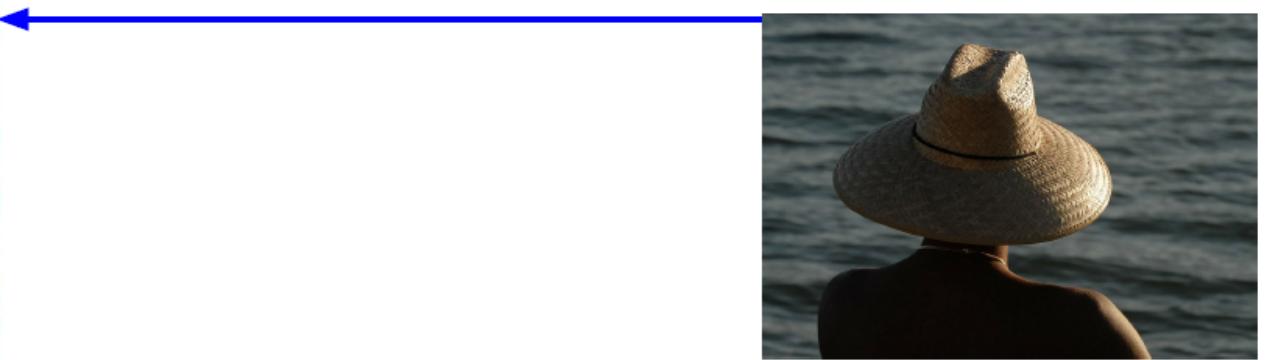


test image

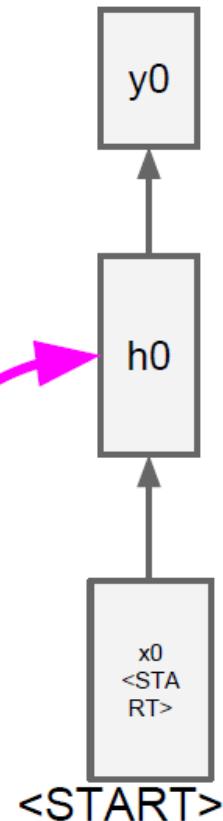




V



test image



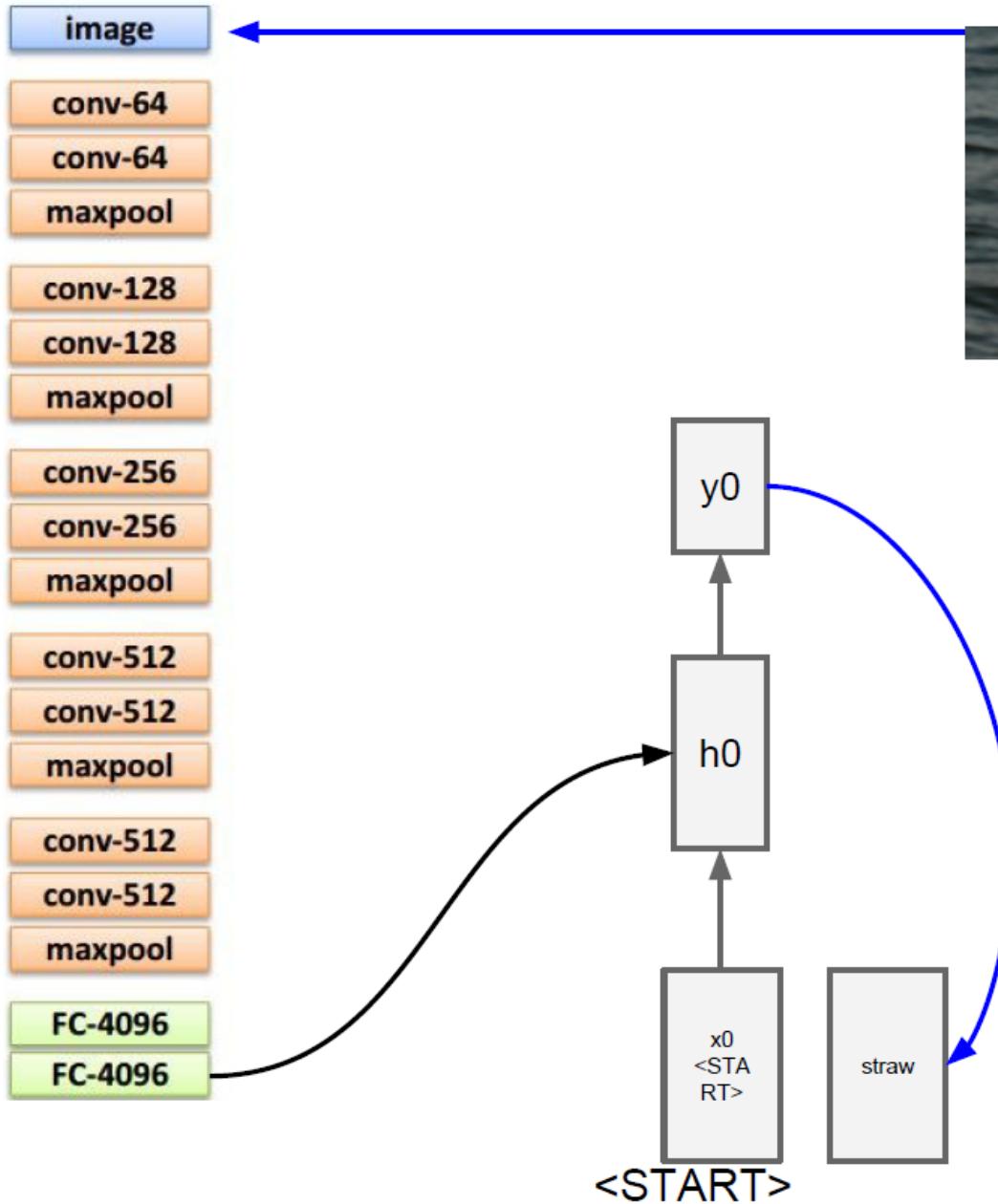
Wih

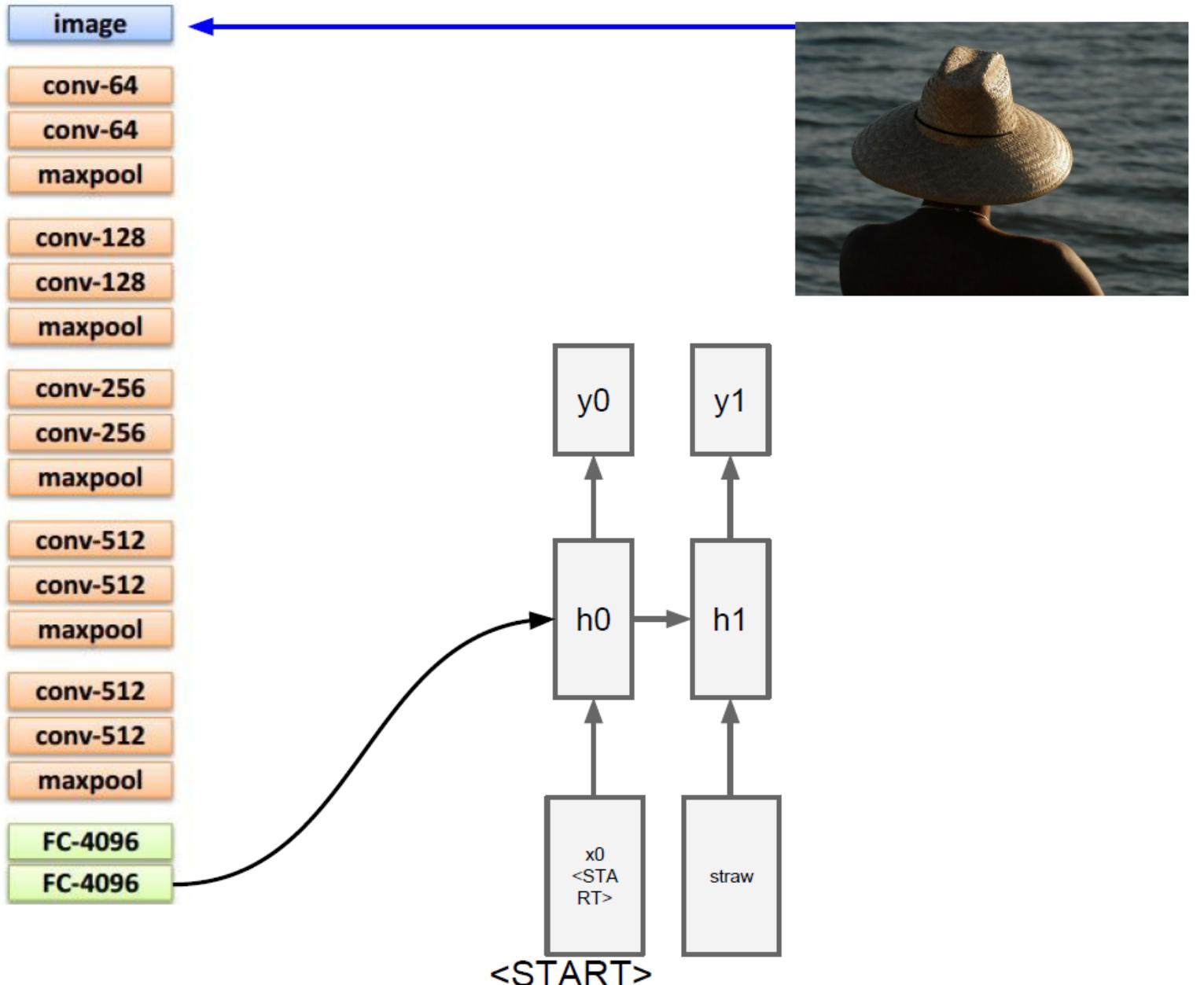
Рекуррентная функция

$$h = \tanh(Wxh * x + Whh * h + Wih * v)$$



test image





test image



image

conv-64

conv-64

maxpool

conv-128

conv-128

maxpool

conv-256

conv-256

maxpool

conv-512

conv-512

maxpool

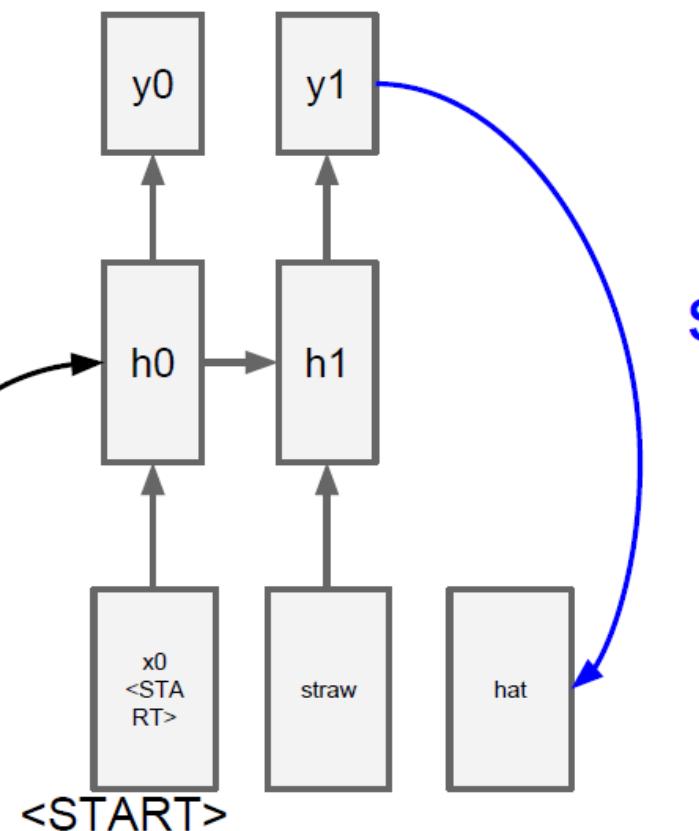
conv-512

conv-512

maxpool

FC-4096

FC-4096

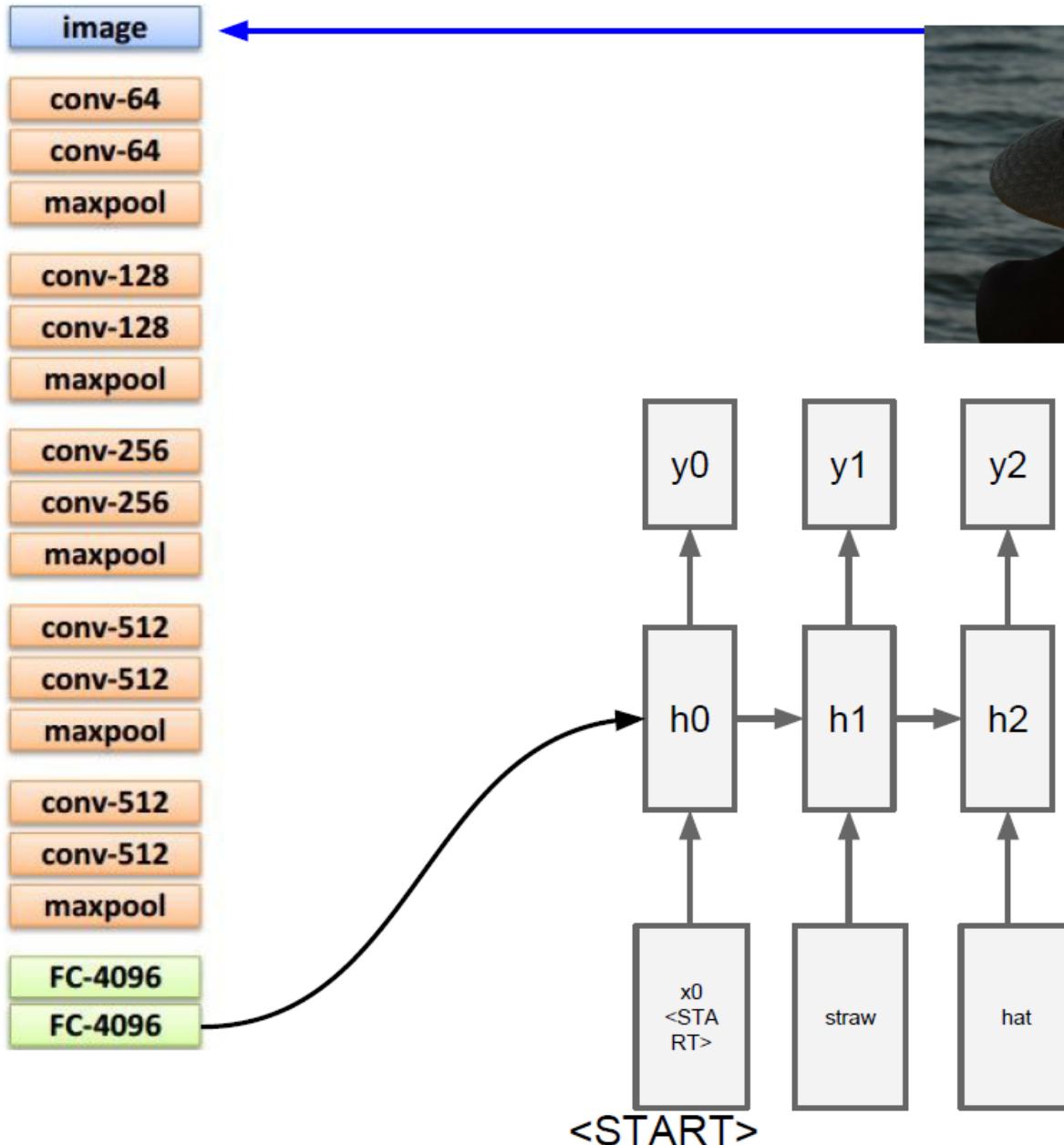


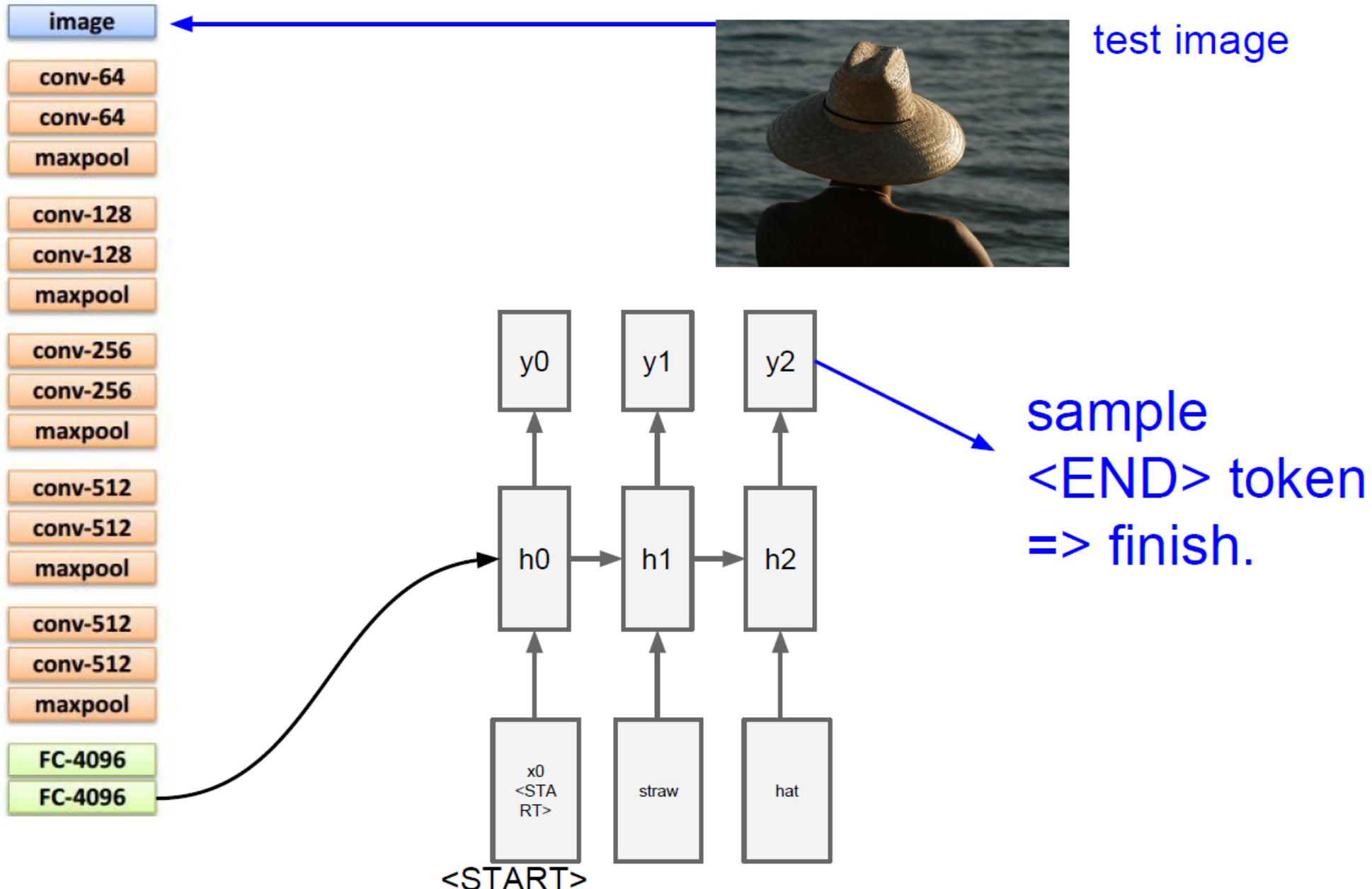
test image

sample!



test image





# Пример работы. Image Captioning



*A cat sitting on a suitcase on the floor*



*A cat is sitting on a tree branch*



*A dog is running in the grass with a frisbee*



*A white teddy bear sitting in the grass*



*Two people walking on the beach with surfboards*



*A tennis player in action on the court*



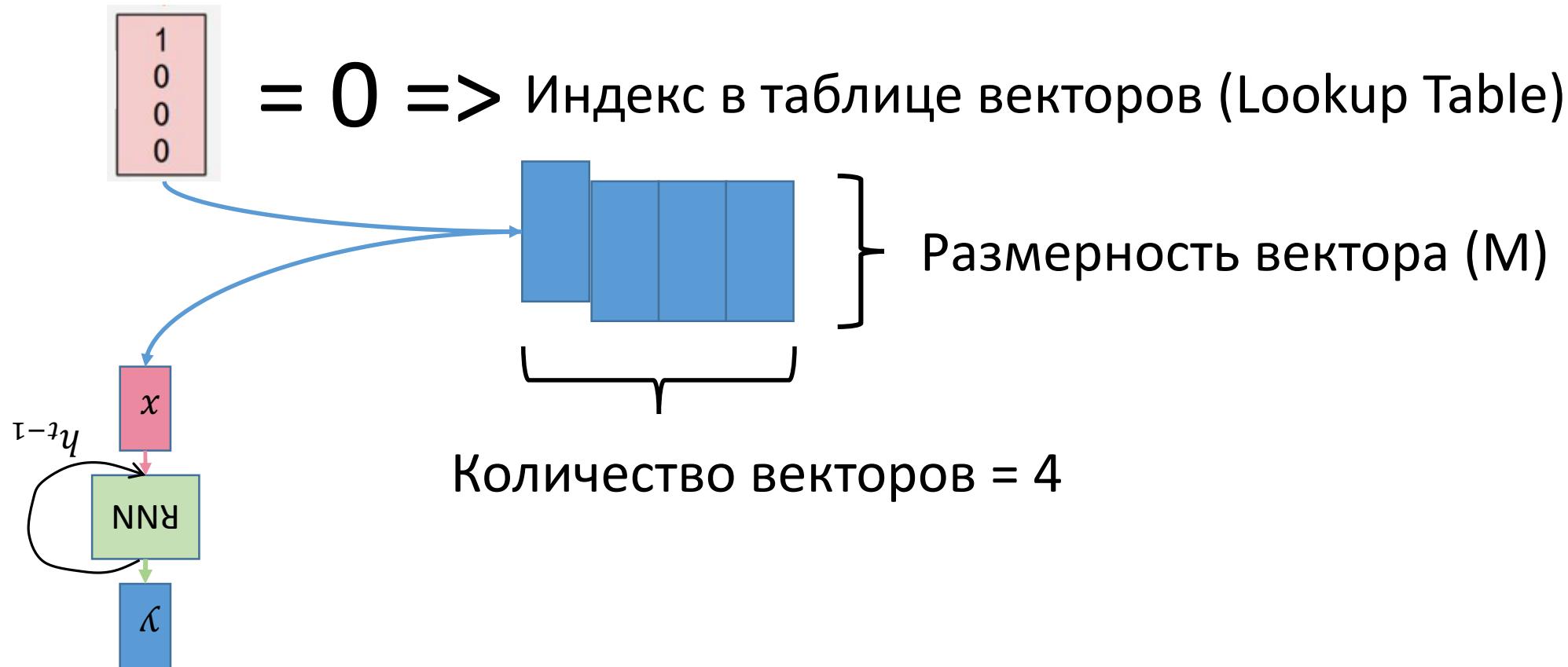
*Two giraffes standing in a grassy field*



*A man riding a dirt bike on a dirt track*

# Практика. Embedding.

- Входной вектор удобно представлять не как one-hot-encoding



# Практика. Embedding.

```
class torch.nn.Embedding(num_embeddings, embedding_dim, padding_idx=None,  
max_norm=None, norm_type=2, scale_grad_by_freq=False, sparse=False) [source] 
```

A simple lookup table that stores embeddings of a fixed dictionary and size.

This module is often used to store word embeddings and retrieve them using indices. The input to the module is a list of indices, and the output is the corresponding word embeddings.

**Parameters:**

- **num\_embeddings** (*int*) – size of the dictionary of embeddings
- **embedding\_dim** (*int*) – the size of each embedding vector
- **padding\_idx** (*int, optional*) – If given, pads the output with zeros whenever it encounters the index.
- **max\_norm** (*float, optional*) – If given, will renormalize the embeddings to always have a norm lesser than this
- **norm\_type** (*float, optional*) – The p of the p-norm to compute for the max\_norm option
- **scale\_grad\_by\_freq** (*boolean, optional*) – if `True`, gradient w.r.t. weight matrix will be a sparse tensor. See Notes for more details regarding sparse gradients.

**Variables:**

**weight** (*Tensor*) – the learnable weights of the module of shape (num\_embeddings, embedding\_dim)

**Shape:**

- Input: LongTensor ( $N, W$ ),  $N$  = mini-batch,  $W$  = number of indices to extract per mini-batch

`class torch.nn.LSTM(*args, **kwargs)` [source]

Applies a multi-layer long short-term memory (LSTM) RNN to an input sequence.

For each element in the input sequence, each layer computes the following function:

$$\begin{aligned} i_t &= \text{sigmoid}(W_{ii}x_t + b_{ii} + W_{hi}h_{(t-1)} + b_{hi}) \\ f_t &= \text{sigmoid}(W_{if}x_t + b_{if} + W_{hf}h_{(t-1)} + b_{hf}) \\ g_t &= \tanh(W_{ig}x_t + b_{ig} + W_{hg}h_{(t-1)} + b_{hg}) \\ o_t &= \text{sigmoid}(W_{io}x_t + b_{io} + W_{ho}h_{(t-1)} + b_{ho}) \\ c_t &= f_t * c_{(t-1)} + i_t * g_t \\ h_t &= o_t * \tanh(c_t) \end{aligned}$$

where  $h_t$  is the hidden state at time  $t$ ,  $c_t$  is the cell state at time  $t$ ,  $x_t$  is the hidden state from the previous layer at time  $t$  or  $\text{input}_t$  for the first layer, and  $i_t$ ,  $f_t$ ,  $g_t$ ,  $o_t$  are the input, forget, cell, and output gates, respectively.

Parameters:

- **input\_size** – The number of expected features in the input  $x$
- **hidden\_size** – The number of features in the hidden state  $h$
- **num\_layers** – Number of recurrent layers.
- **bias** – If `False`, then the layer does not use bias weights  $b_{\text{ih}}$  and  $b_{\text{hh}}$ . Default: `True`

- **batch\_first** – If `True`, then the input and output tensors are provided as (batch, seq, feature)
- **dropout** – If non-zero, introduces a dropout layer on the outputs of each RNN layer except the last layer
- **bidirectional** – If `True`, becomes a bidirectional RNN. Default: `False`

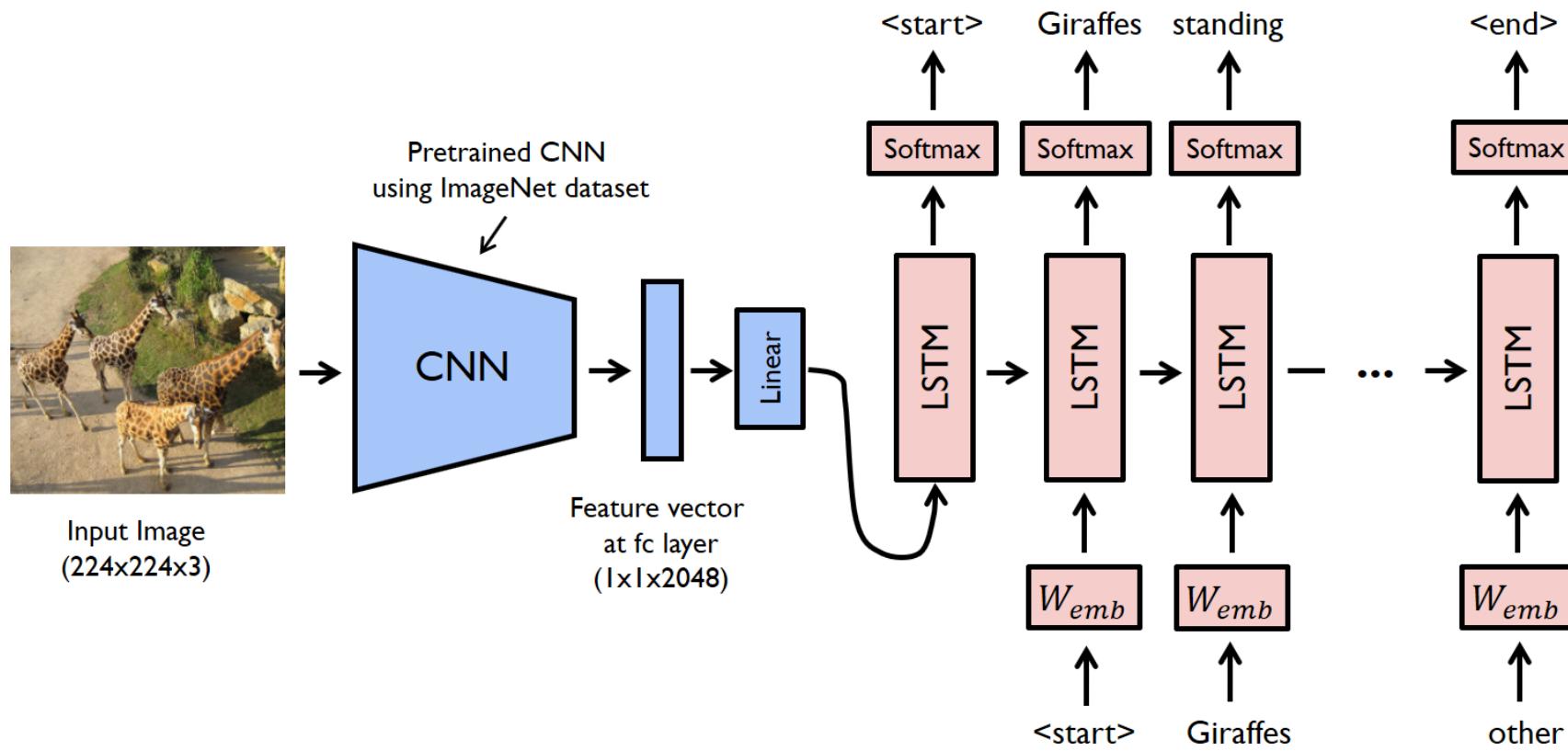
Inputs:  $\text{input}, (\text{h}_0, \text{c}_0)$

- **input** (seq\_len, batch, input\_size): tensor containing the features of the input sequence. The input can also be a packed variable length sequence. See [`torch.nn.utils.rnn.pack\_padded\_sequence\(\)`](#) for details.
- **h\_0** (num\_layers \* num\_directions, batch, hidden\_size): tensor containing the initial hidden state for each element in the batch.
- **c\_0** (num\_layers \* num\_directions, batch, hidden\_size): tensor containing the initial cell state for each element in the batch.

Outputs:  $\text{output}, (\text{h}_n, \text{c}_n)$

- **output** (seq\_len, batch, hidden\_size \* num\_directions): tensor containing the output features ( $h_t$ ) from the last layer of the RNN, for each  $t$ . If a [`torch.nn.utils.rnn.PackedSequence`](#) has been given as the input, the output will also be a packed sequence.
- **h\_n** (num\_layers \* num\_directions, batch, hidden\_size): tensor containing the hidden state for  $t=\text{seq\_len}$
- **c\_n** (num\_layers \* num\_directions, batch, hidden\_size): tensor containing the cell state for  $t=\text{seq\_len}$

# Практика. Image Captioning



# Перерыв и потом LSTM

# Long Short Term Memory (LSTM)

**Vanilla RNN**

$$h_t = \tanh \left( W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \right)$$

**LSTM**

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

Hochreiter and Schmidhuber, “Long Short Term Memory”,  
Neural Computation  
1997

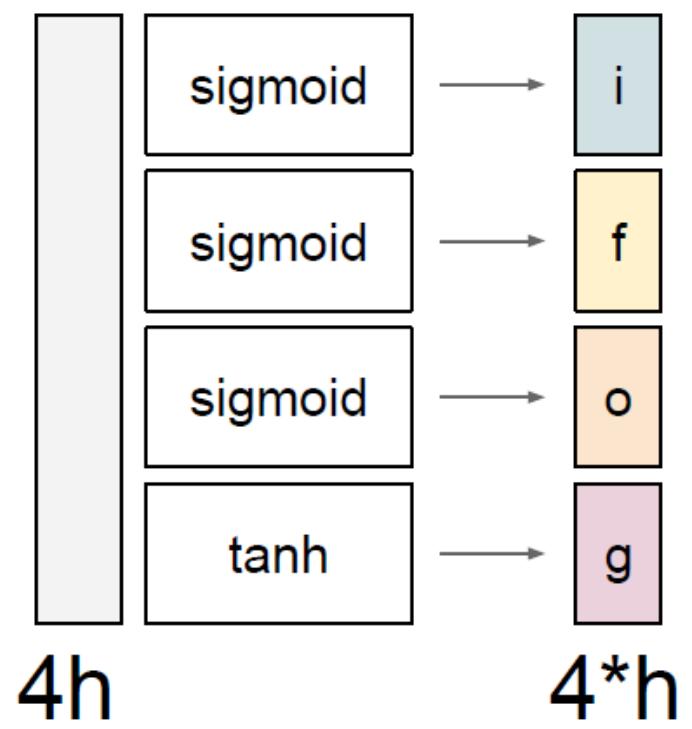
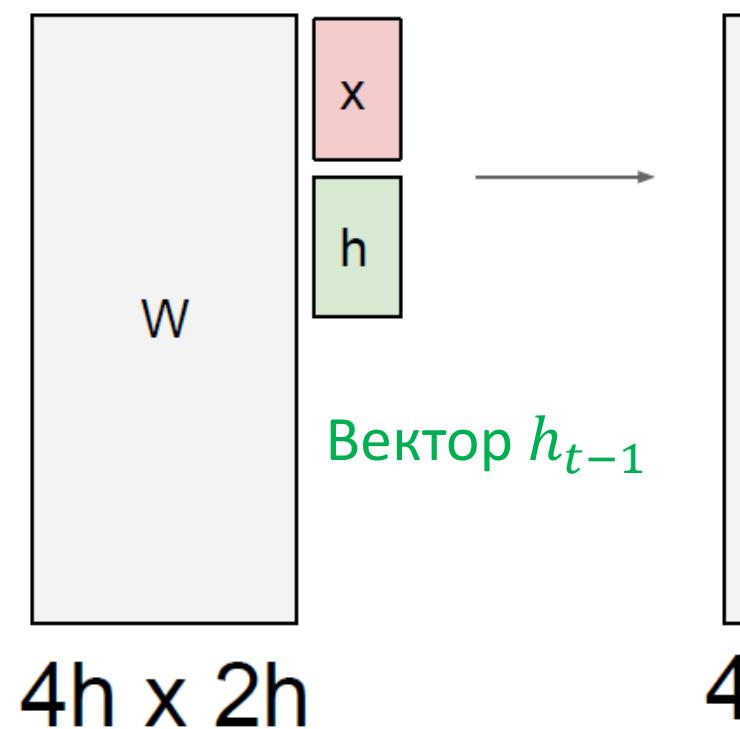
# Long Short Term Memory (LSTM)

$$\begin{aligned}\mathbf{c}'_t &= \tanh(W_{xc} \mathbf{x}_t + W_{hc} \mathbf{h}_{t-1} + \mathbf{b}_{c'}) && \textit{candidate cell state} \\ \mathbf{i}_t &= \sigma(W_{xi} \mathbf{x}_t + W_{hi} \mathbf{h}_{t-1} + \mathbf{b}_i) && \textit{input gate} \\ \mathbf{f}_t &= \sigma(W_{xf} \mathbf{x}_t + W_{hf} \mathbf{h}_{t-1} + \mathbf{b}_f) && \textit{forget gate} \\ \mathbf{o}_t &= \sigma(W_{xo} \mathbf{x}_t + W_{ho} \mathbf{h}_{t-1} + \mathbf{b}_o) && \textit{output gate} \\ \mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \mathbf{c}'_t, && \textit{cell state} \\ \mathbf{h}_t &= \mathbf{o}_t \odot \tanh(\mathbf{c}_t) && \textit{block output}\end{aligned}$$

$$y_t = W_{hy} h_t$$

# Long Short Term Memory (LSTM)

Вектор  $x$



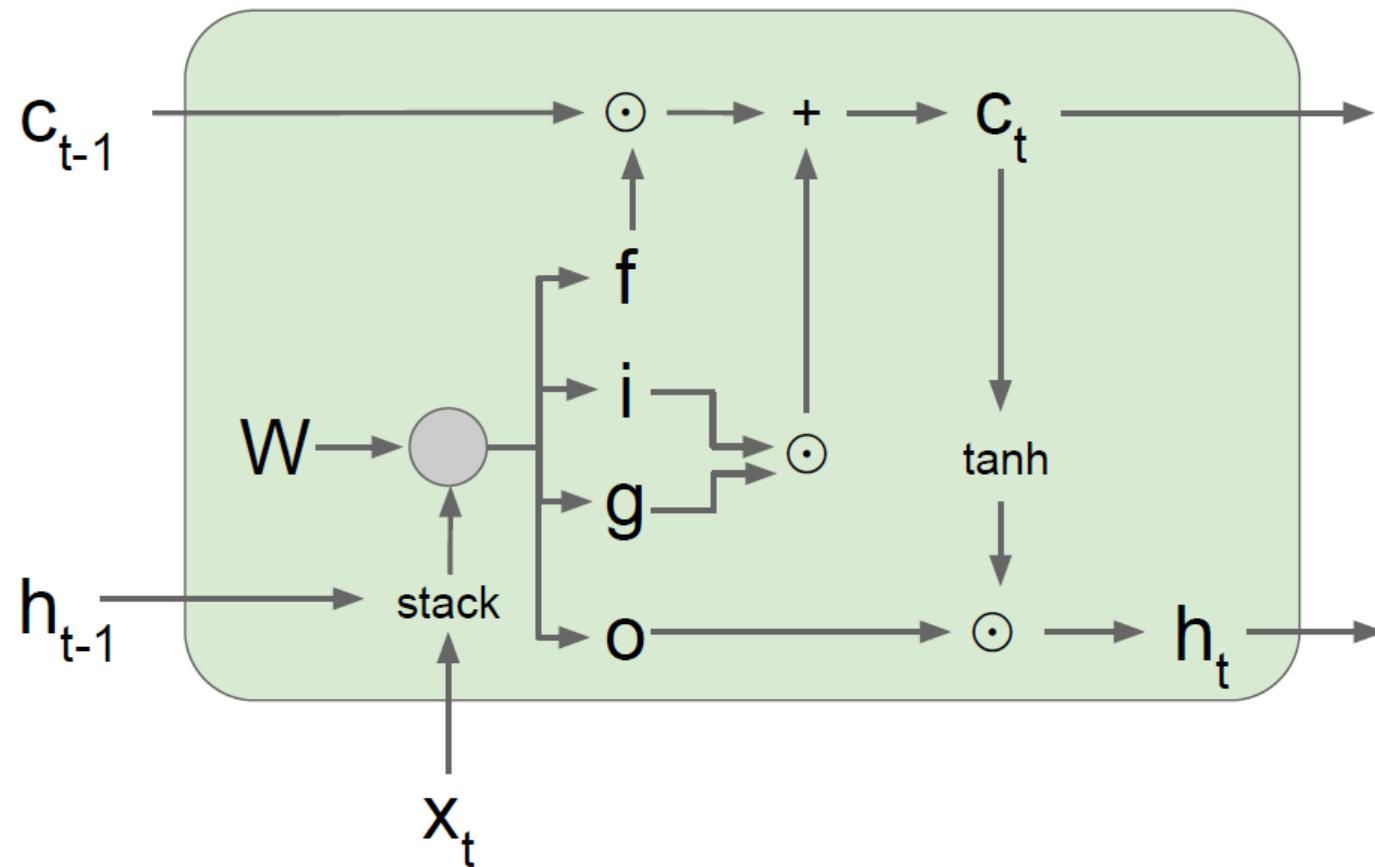
- f: Forget gate, Забывать ли состояние элемента
- i: Input gate, Писать ли состояние элемента
- g: Gate gate, Сколько писать в элемент
- o: Output gate, Сколько выводить из элемента

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

# Long Short Term Memory (LSTM)



$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$
$$c_t = f \odot c_{t-1} + i \odot g$$
$$h_t = o \odot \tanh(c_t)$$

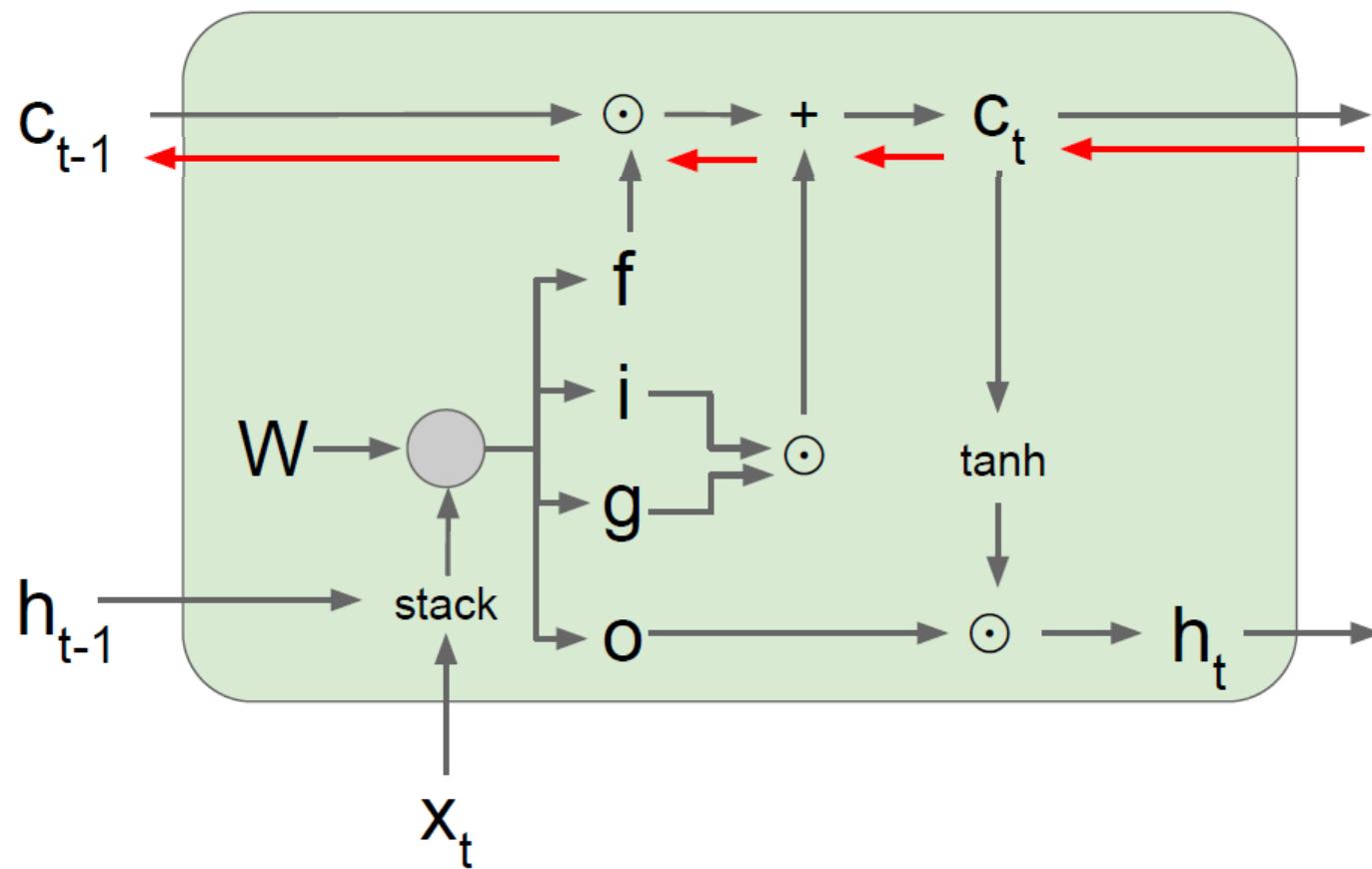
# LSTM. Gradient Flow

- Что будет если  $\text{forget\_gate}=1$  всегда?
- Как надо инициализировать  $\text{bias\_forget\_gate}$ ?

$$\mathbf{c}_t = \mathbf{c}_{t-1} + i_t \odot \mathbf{c}'_t.$$

$$\frac{\partial \mathbf{c}_t}{\partial \mathbf{c}_{t-1}} = 1.$$

# LSTM. Gradient Flow



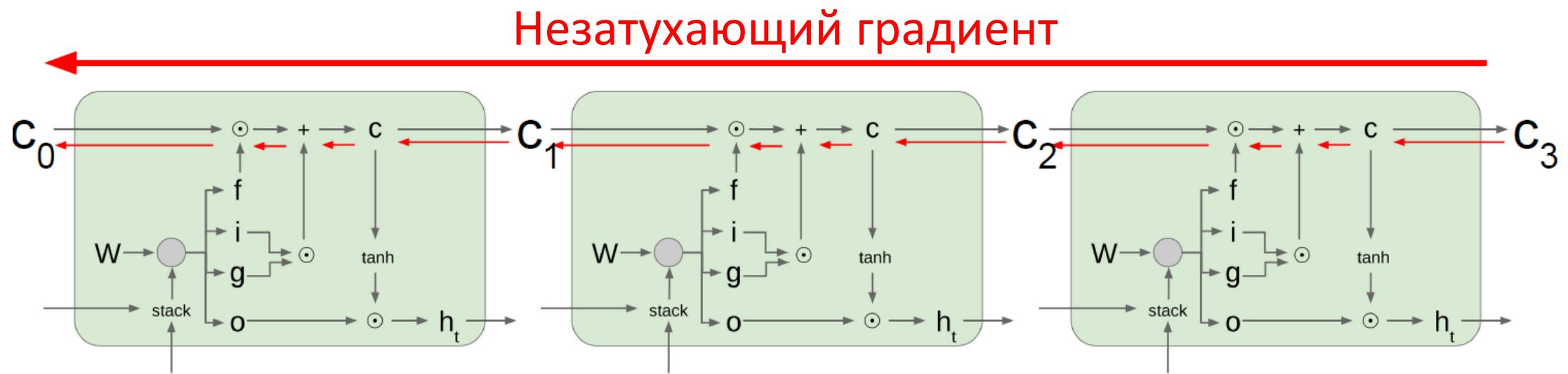
Обратный проход от  $c_t$  до  $c_{t-1}$  перемножаются только с  $f$  без матрицы  $W$

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

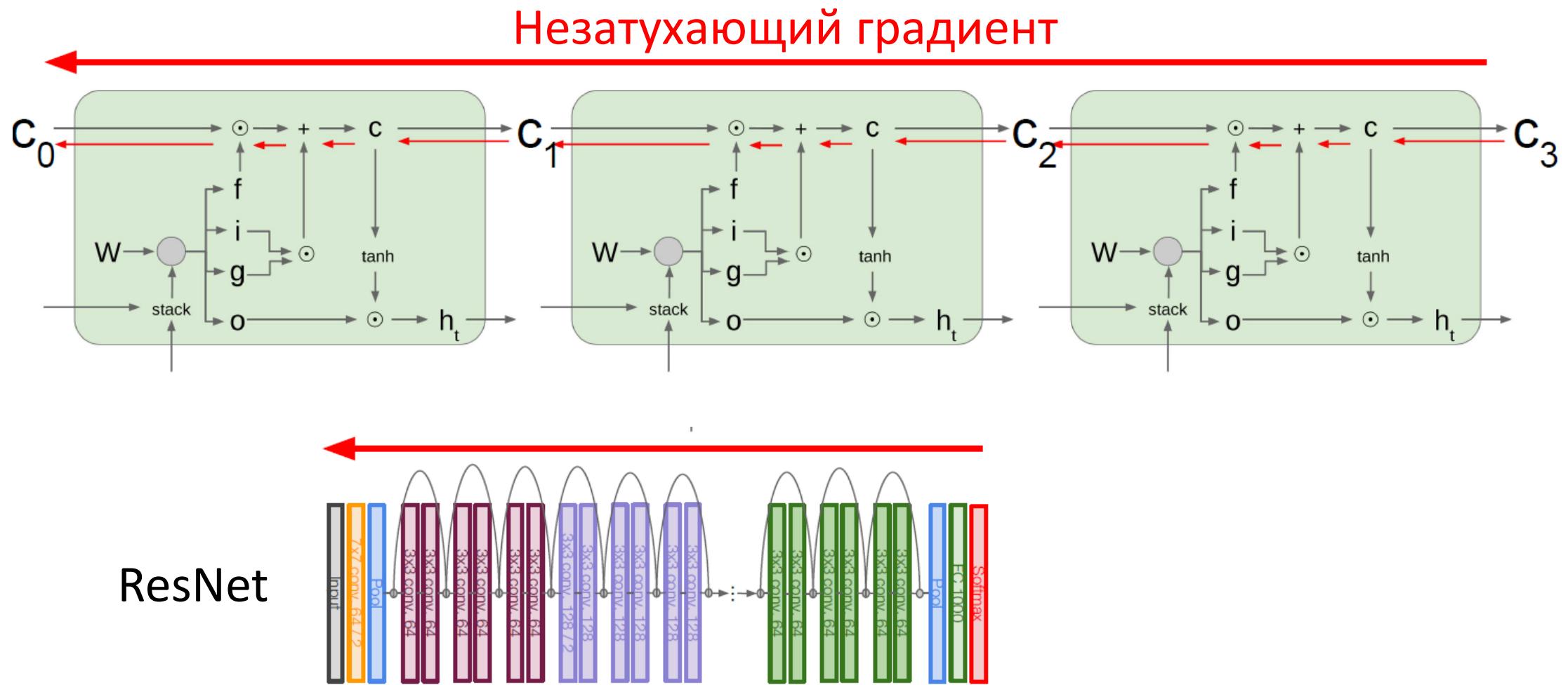
$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

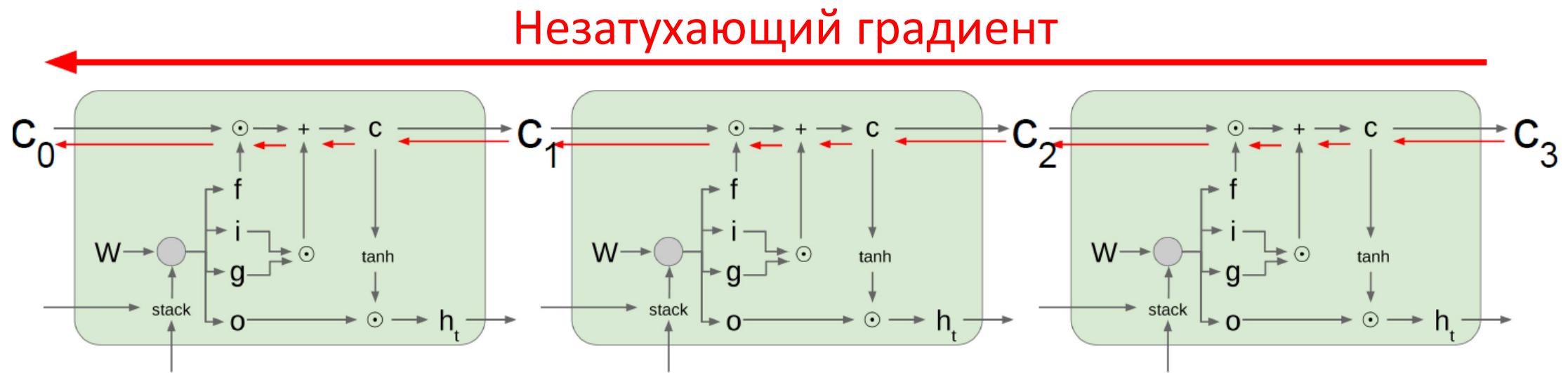
# LSTM. Gradient Flow



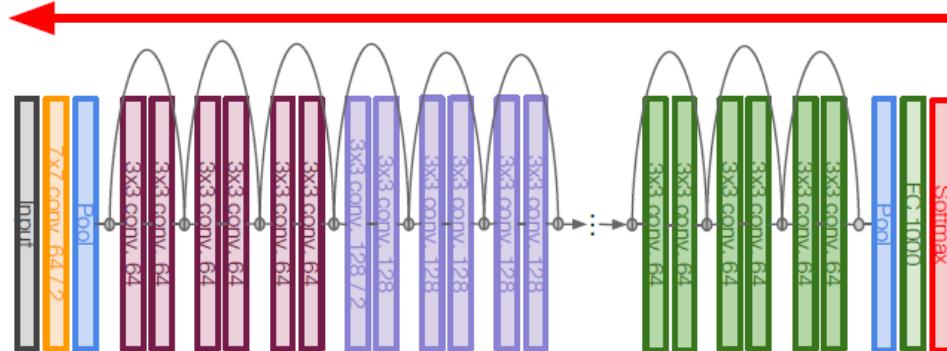
# LSTM. Gradient Flow



# LSTM. Gradient Flow



ResNet



**Highway Networks**

$$g = T(x, W_T)$$

$$y = g \odot H(x, W_H) + (1 - g) \odot x$$

Srivastava et al, "Highway Networks",  
ICML DL Workshop 2015

# Long Short Term Memory (LSTM)

$$\begin{aligned}\mathbf{c}'_t &= \tanh(W_{xc} \mathbf{x}_t + W_{hc} \mathbf{h}_{t-1} + \mathbf{b}_{c'}) && \textit{candidate cell state} \\ \mathbf{i}_t &= \sigma(W_{xi} \mathbf{x}_t + W_{hi} \mathbf{h}_{t-1} + \mathbf{b}_i) && \textit{input gate} \\ \mathbf{f}_t &= \sigma(W_{xf} \mathbf{x}_t + W_{hf} \mathbf{h}_{t-1} + \mathbf{b}_f) && \textit{forget gate} \\ \mathbf{o}_t &= \sigma(W_{xo} \mathbf{x}_t + W_{ho} \mathbf{h}_{t-1} + \mathbf{b}_o) && \textit{output gate} \\ \mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \mathbf{c}'_t, && \textit{cell state} \\ \mathbf{h}_t &= \mathbf{o}_t \odot \tanh(\mathbf{c}_t) && \textit{block output}\end{aligned}$$

$$y_t = W_{hy} h_t$$

# LSTM – peepholes («замочные скважины»)

- Зависят ли gate от состояния ячейки памяти (cell)?

$$i_t = \sigma (W_{xi}x_t + W_{hi}h_{t-1} + W_{pi}c_{t-1} + b_i)$$

$$f_t = \sigma (W_{xf}x_t + W_{hf}h_{t-1} + W_{pf}c_{t-1} + b_f)$$

$$o_t = \sigma (W_{xo}x_t + W_{ho}h_{t-1} + W_{po}c_{t-1} + b_o)$$

# Vanilla RNN – как ее получить из LSTM?

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

$$y_t = W_{hy} h_t$$

# LSTM – важные моменты

- Внутренняя ячейка памяти – cell
- Забывающий гейт (forget gate)
- Выходной гейт (output gate)
- Незатухающая ошибка по памяти
- Функция активации выхода не дает градиентам

# GRU RNN

**GRU** [*Learning phrase representations using rnn encoder-decoder for statistical machine translation*, Cho et al. 2014]

$$\mathbf{u}_t = \sigma(W_{xu} \mathbf{x}_t + W_{hu} \mathbf{h}_{t-1} + \mathbf{b}_u),$$

$$\mathbf{r}_t = \sigma(W_{xr} \mathbf{x}_t + W_{hr} \mathbf{h}_{t-1} + \mathbf{b}_r),$$

$$\mathbf{h}'_t = \tanh(W_{xh'} \mathbf{x}_t + W_{hh'} (\mathbf{r}_t \odot \mathbf{h}_{t-1})),$$

$$\mathbf{h}_t = (1 - \mathbf{u}_t) \odot \mathbf{h}'_t + \mathbf{u}_t \odot \mathbf{h}_{t-1}.$$

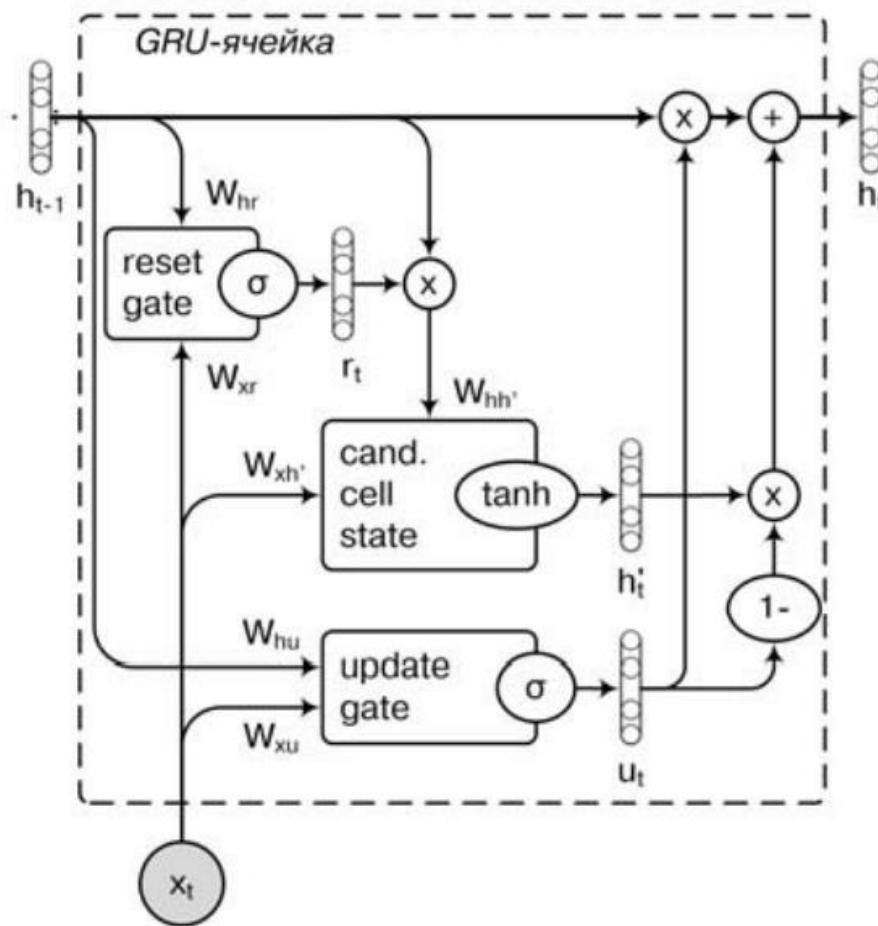
# GRU – gated recurrent unit

- Совмещаем выходные и забывающий гейты – их заменяет `update_gate (u)`
- Совмещаем `hidden_state(h)` и `cell_state(c)` – объединяем их
- Добавляем `reset_gate`

# GRU – gated recurrent unit

- reset\_gate – определяет как объединить новый вход ( $x$ ) с имеющимся состоянием ( $h$ )
- update\_gate – какую часть памяти оставить неизменной
- hiddent\_state опять имеет линейную связь для градиентов через update\_gate

# GRU – gated recurrent unit



# Vanilla RNN – как ее получить из GRU?

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

$$y_t = W_{hy} h_t$$

# Использованные материалы

- Книга «**Николенко, Кадурин, Архангельская: Глубокое обучение. Погружение в мир нейронных сетей**»